0-315-03673-7

National Library of Canada
Canadian Theses Division
Ottawa, Canada
K1A 0N4

Bibliothèque nationale du Canada
Division des thèses canadiennes

51568

PERMISSION TO MICROFILM — AUTORISATION DE MICROFILMER

- Please print or type — Écrire en lettres moulées ou dactylographier

Full Name of Author — Nom complet de l'auteur
IAN GRAHAM REAVILL

Date of Birth — Date de naissance
JULY 23, 1954

Country of Birth — Lieu de naissance
TRINADAD

Permanent Address — Résidence fixe
11735 91 Ave
EDMONTON ALBERTA
T6G 1B1

Title of Thesis — Titre de la thèse
A Software System for Distributed Process Control.

University — Université
University of Alberta.

Degree for which thesis was presented — Grade pour lequel cette thèse fut présentée
Master of Science

Year this degree conferred — Année d'obtention de ce grade
1981

Name of Supervisor — Nom du directeur de thèse
DR. D.G. Fisher

Permission is hereby granted to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

L'autorisation est, par la présente, accordée à la BIBLIOTHÈQUE NATIONALE DU CANADA de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans l'autorisation écrite de l'auteur.

Date
APRIL 2, 1981

Signature
Ian Reavill

NL-91 (4/77)

National Library of Canada
Collections Development Branch

Canadian Theses on
Microfiche Service

Bibliothèque nationale du Canada
Direction du développement des collections

Service des thèses canadiennes
sur microfiche

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us a poor photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de mauvaise qualité.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

Ottawa, Canada
K1A 0N4

NL-339 (Rev. 8/80)

THE UNIVERSITY OF ALBERTA

A Software System for Distributed Process Control

by

© Ian G. Reavill

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE
OF Master of Science
IN
Process Control

Department of Chemical Engineering

EDMONTON, ALBERTA
Spring, 1981

THE UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR Ian G. Reavill

TITLE OF THESIS A Software System for Distributed Process Control

DEGREE FOR WHICH THESIS WAS PRESENTED Master of Science

YEAR THIS DEGREE GRANTED Spring, 1981

Permission is hereby granted to THE UNIVERSITY OF ALBERTA LIBRARY to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

(SIGNED) Ian Reavill

PERMANENT ADDRESS:

11735 91 Ave
Edmonton Alberta
T6G 1B1

DATED April 2 1981

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled A Software System for Distributed Process Control submitted by Ian G. Reavill in partial fulfilment of the requirements for the degree of Master of Science in Chemical Engineering.

.......D.G. Fisher.......
Supervisor

....

.......P.K. Leung...

....

Date..November 3, 1980..........

Abstract

The Distributed Supervisory and Control (DISCO) program is part of an application package being developed in the Department of Chemical Engineering at the University of Alberta to evaluate, and demonstrate the various alternatives for industrial control. The second phase of the development, completed as part of this thesis, investigated the areas of system integrity and distributed processing.

Computers are playing an increasing role in the control of large industrial plants and hence the "integrity" of the system is also of increasing importance. The need for data base integrity and the method of recovery after a system failure in a process control environment were investigated and compared to the needs and solutions used in typical data processing applications. The solution implemented used formal integrity measures of the type used in data processing applications only where necessary. It also took advantage of the characteristic that much of the data was short lived and thus needed minimal integrity measures. This dual type of function for integrity was necessary to maintain a constant time base while still performing process control.

Distributed computing offers a number of advantages compared to a centralized system. The advantages and disadvantages were analysed in terms of the benefits gained in a process control application. A critical feature of a distributed control system is the communication mechanism

since it ties the individual processors together. The software protocols that ensured consistancy and prevented deadlocks were quite simple compared to the data processing applications. This was possible because the volume of data communicated between processors in a process control application is relatively small.

The completion of the work of this thesis leaves the DISCO program closer to a production type package usable in a process control environment. In addition, the package can now be used to investigate the areas of load leveling over the network and controlled degeneration in the event of a system failure.

Acknowledgements

The author wishes to express his sincere gratitude to Dr. D.G. Fisher for his assistance, guidance and encouragement during this thesis.

My parents help in performing the tedious task of proof reading was gratefully appreciated. Their assistance as a sounding board for ideas was also a great help.

An acknowledgement must be extended to the staff (past and present) of the Data Acquisition, Control and Simulation Centre for the generous help.

Financial support by the National Sciences and Engineering Research Council was also appreciated.

Table of Contents

Chapter Page

1. Introduction 1
    - 1.1 Definition of a Process Control Application Program 1
    - 1.2 Development Phases in the Process Control Application Program 2
        - 1.2.1 DISCO 4
        - 1.2.2 Integrity and Distributed Control; Work of this thesis 8
2. Characteristics of Process Control 11
    - 2.1 Structure of the Control Problem 11
    - 2.2 Characterization of the Process Control Application 18
3. System Reliability 21
    - 3.1 Data Integrity 23
        - 3.1.1 Erroneous Updates 23
        - 3.1.2 Erasing of data 25
    - 3.2 Comparison of Requirements 25
        - 3.2.1 Process Control 26
        - 3.2.2 Data Processing 26
    - 3.3 Transactional Analysis 28
        - 3.3.1 Locks 32
            - 3.3.1.1 Granularity of Locks 33
            - 3.3.1.2 Consistency 35
            - 3.3.1.3 Deadlocks 35
    - 3.4 Recovery 38
    - 3.5 Application Integrity 42
        - 3.5.1 Restore versus Restart 43

4. Distributed Systems 45

4.1 Communications Requirements 45

4.1.1 Process Requirements 46

4.1.2 Process Applications in a Multiprocessor Environment 46

4.1.2.1 Cascade Control 46

4.1.2.2 Signal Conditioning 47

4.1.2.3 Reporting 48

4.1.3 Comparisom of Process Control with Data Processing 49

4.2 Communication Program 50

4.2.1 H.P. Communications Package 50

4.2.2 Additions to the Communication Package ...51

5. Design of System 56

5.1 Module Description 56

5.2 Flow of Process Control Application in Disco 60

5.3 Detailed Design 64

5.3.1 ACCES 65

5.3.1.1 Modules 66

5.3.2 COM 71

5.3.2.1 COM Modules 72

6. Program Testing 79

6.1 Operability Testing 80

6.1.1 Functional level 80

6.1.2 Program Testing 81

6.1.3 Application Testing 82

6.2 Performance Evaluation 84

7. Summary, Conclusions and Future Work 89

7.1 Contributions of the Thesis 89

7.2 Conclusions and Recommendations 90

7.3 Future Work 93

7.3.1 Future Modules 93

7.3.2 Future Work 94

Bibliography 96

Appendix A: Alarm Processing Segment...................101

A.1 Discussion of Functions 102

A.1.1 Identification 102

A.1.2 Action 102

A.2 Design of the Module 103

A.2.1 Identification 103

A.2.2 Action 105

A.3 Discussion of Design 106

Appendix B: Advantages and Disadvantages of Using a Distributed Computer Network 107

B.1 Advantages of Using a Distributed System 107

B.1.1 Reliability, Recoverability and Stability 107

B.1.2 Communications 109

B.1.3 Economy of Dedicated Processors 110

B.1.4 Incremental Growth and Flexibility 111

B.1.5 Installation 111

B.2 Disadvantages Connected With a Distributed System 112

B.2.1 Performance 112

B.2.1.1 Response Time 112

B.2.1.2 Skew 114

B.2.2 Duplication of Function 117

Table of Figures

1. Figure 1.1 DISCO Data Structure........................8

2. Figure 4.1 Protocol for Communication................53

3. Figure 4.2 Comparison of Secure Write and
Send Transaction..........................54

4. Figure 5.1 Module Layout..............................59

5. Figure 5.2 Sequence of Input Activity................62

6. Figure 5.3 Sequence of Control Activity..............63

7. Figure 5.4 Sequence of Output Activity...............64

8. Figure 5.5 Detailed Module Layout of ACCES...........67

9. Figure 5.6 Logic of ACCES Module.....................68

10. Figure 5.7 Detailed Module Layout of COM............73

11. Figure 5.8 Logic of MCOM module of COM..............74

12. Figure 5.9 Logic of RCOM module of COM..............75

13. Figure A.1 Communication String Layout for Alarm
Segment....................................105

14. Figure B.1 String Network...........................113

15. Figure B.2 Fully Connected Network.................113

16. Figure B.3 Example of Communication Delay Creating
Skew.................................116

Table of Tables

1. Table 1.1 Objectives for Process Control Program......3
2. Table 1.2 Constraints on Application Program..........4
3. Table 1.3 Segment Library............................7
4. Table 5.1 Module Functions and Development Stage......................60
5. Table 6.1 Timing Results of ACCES Functions...........87
6. Table 6.2 Timing Results of COM Functions.............87

1. INTRODUCTION

The diminishing supply of raw materials presents a tough challenge for today's Chemical Engineer. This challenge appears in two forms; firstly, to maximize the efficiency of the existing processing plants and secondly, to devise new processes that are more efficient or produce a product that can replace existing materials at a cheaper cost. The first objective results in the existing plant being run closer to constraints. The second results in complex control strategies to match the additional complexity in the production of the better product. In addition, the cost of manpower is increasing so there is a great pressure to increase the productivity of the existing operators. All of the objectives may only be met through the increased use of computers. The topic of this thesis is a continuation of the development of a package that enables a computer to be used as a tool for process plant control. The particular areas of study are the overall reliabilty of the hardware and software package and the implications of a multiprocessor computer network to a control scheme.

1.1 Definition of a Process Control Application Program

A computer controlling a plant is also the interface between the process and the people who perform the day to day operation, plan plant operation and maintain the equipment. The process control application program simplifies the use of the computer as a tool by these people during the performance of their various jobs. In addition, it also performs and monitors the actual control algorithms that adjust the control variables. The success of the application program is determined by its ability to provide the functions needed by these people and by the control schemes.

1.2 Development Phases in the Process Control Application Program

The objectives listed in Table 1.1, define the characteristics the process control application program must provide. The implementation of the program is executed in a number of phases.

The development of the process control application program is subject to constraints set by the general design specification and placed by the hardware the program is implemented on. These are summarized in Table 1.2.

1.2.1 DISCO

The initial phase was the work of Andrew Brenek (6) and dealt entirely with the implementation of modules for process control on one processor. The program, DISCO, set up modules that would be run every second. Considerable work was done to ensure a direct correlation between the real time clock and the number of executions of the subprogram.

Objectives of a Process Control System Program

1. To operate in Real-time
2. To be Reliable
3. To be Transparent to the User
4. To be Flexible
5. To be Cost Effective

Table 1.1 Objectives of a Process Control Program

Control Constraints

1) Minimum execution period is 1 second
    - required for DDC loops performing regulatory control

System Constraints

2) All real-time aspects of the application program will run under the DISCO executive

3) Application program (DISCO) to be written in a high level language
    - for maintainability

4) Application program (DISCO) to use vendor's communication package
    - for maintainability

5) DISCO to run in a multi-program environment
    - other programs will be running during idle periods

Hardware Constraints

6) Program size limited to 28 pages

Table 1.2 Constraints on the Application Program

The program itself was constructed on the basis that all the pertinent information for any control scheme would be entered in a database. By making one complete cycle through the table all the control problems would be checked and outputs recalculated, status changed, etc. if necessary. Thus all the control tasks of the plant could be executed once every second.

A key consideration is the actual structure of the database. The control task was broken up into sections and subsections with a section or "activity" representing a particular control scheme and a subsection or "segment" performing a particular function required by the control scheme. An activity would be constructed by combining a number of segments in a particular order to generate the control scheme required. The execution of an activity would result in the interpretation of the entries in the table as addresses of the I/O information, tuning constants, or status flags according to the layout stated in the documentation of each segment. Such a table driven processor provides maximum flexibility while still limiting the overhead since the frequently changed values are entries in a table. Various combinations of segments provide a large number of options for a control scheme whereas the actual number of programs is reduced to a few.

The segment processors are the modules where actual computation for the control scheme is performed. The functions the segment processors perform include the retrieval of the raw data from the input points, the calculation of the control action required and the transmittance of the calculated values to their output destinations. The library of the available segments and their functions is in Table 1.3. New segment processors will be written as they are required. In the implementation of a segment processor great emphasis is placed on the documentation since each processor is used by a variety of users.

The layout of the data structures in DISCO are shown in Figure 1.1.

Along with the program that actually implemented process control Andrew Brenek (6) wrote all the utilities that built and loaded the database.

1.2.2 Integrity and Distributed Control; Work of this thesis

This thesis looks at two areas required in the DISCO program in continuing its evolution towards a viable tool for process control in an industrial environment. The first area is the problem of ensuring the reliability of the control schemes against computer hardware and software faults. The second is the need to accomodate a control scheme that is divided into sections running in a number of CPU's. Additions in both of these areas result in extra overhead that must be justified by better performance of the control schemes.

In examining the first area one must consider the types

	Segment Title	Segment Function
1)	Input Segment	- get input data - buffer area for process data
2)	Output Segment	- buffer area for output data - send data to process
3)	Data Accum.	- store data value in array
4)	Filter Segment	- simple exponential filter
5)	Control Seg.	- PID control algorithm
6)	Thermocouple Input	- Input segment for thermocouples
7)	Input Conversion	- convert raw data to eng. units
8)	Sampled-Data Control	- sampled-data control algorithm
9)	Program Scheduler	- schedule supervisory programs
10)	Cascade Control	- two input cascade algorithm
11)	Reverse Data Accum.	- output data stored for test runs
12)	Alarm	- detects and reacts to abnormal conditions

Table 1.3 Segment Library

Data Base

- Activity Record Table

Group

- performs all control of one type
- performs control of a unit (complex)
- made up of 1) Group Header
  2) Activities

Activity

- performs a control loop or problem
- made up of 1) Activity Header
  2) Segments

Segments

- performs specific function (calculation)
- made up of 1) Segment Header
  2) Parameters (real, integer)
  3) Memory area

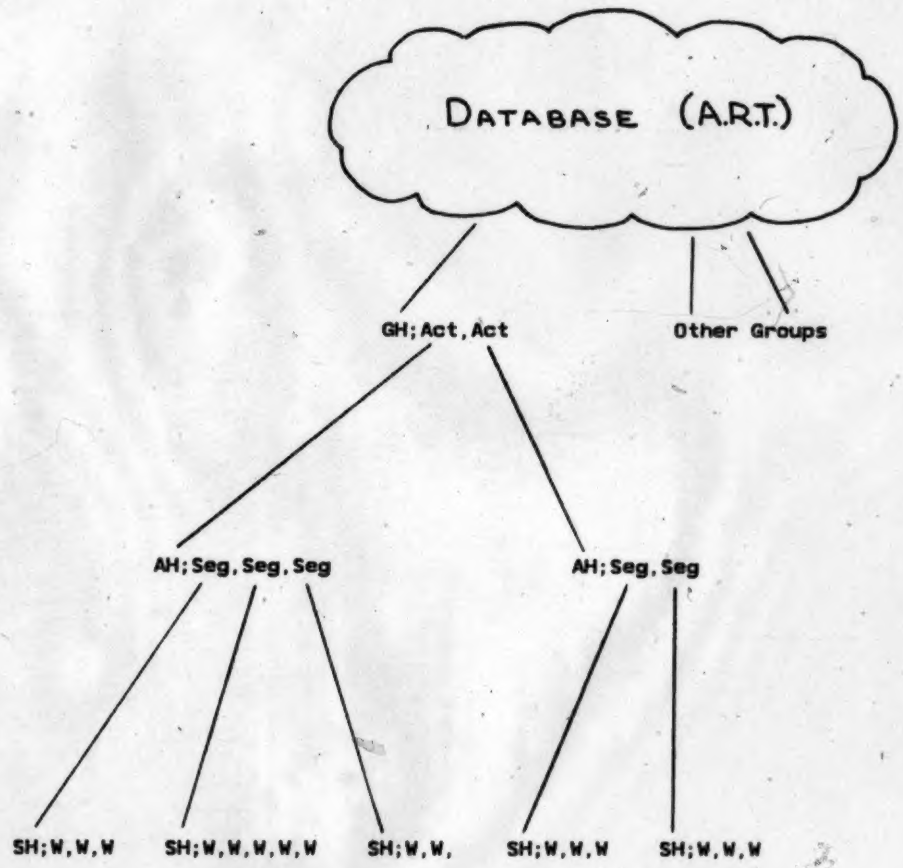
DATABASE (A.R.T.)
GH;Act,Act
Other Groups
AH;Seg,Seg,Seg
AH;Seg,Seg
SH;W,W,W
SH;W,W,W,W,W
SH;W,W,
SH;W,W,W
SH;W,W,W

Figure 1.1 DISCO Data Structure

of computer faults that will effect the control scheme. The typical case which only a back-up computer can totally solve is the complete break-down of the main-frame. It is often hard to justify the cost of an entirely redundant computer that is only used on rare ocassions. Also most processes can stand being on manual control for a number of minutes while the computer is repaired. The key point with such a failure is the recovery process that must occur in order to return the computer system and the process to the normal operating mode. In order to be able to recover, a procedure is required to moniter the application at all times so that the condition of all the process loops is always known. The second requirement is a method that enables the computer to return to an appropriate state for resumption of normal operation after a failure occurs. Some of the problems encountered are similar to those encountered in data processing application such as banking systems where reliability is also important. Process control puts stringent limits on the response time to attain this reliability since a consistent time base must be maintained.

The second area of interest of this thesis is the implementation of a process control strategy on a distributed network of processors. A network is used to reduce installation costs and dependency on a single computer. A major problem resulting from this evolution of process control is the communication of the condition of each processor to all the other processors in the network.

Another key to the operation of such a network is the efficiency and reliability of the communication between processors. However, a process control system is usually based in small minicomputers which lack the resources that the large processors possess.

In the implementation of this enhancement of the DISCO program any trade-offs and their ramifications will be examined as the design proceeds. In addition the program structure will be modular to achieve all the benefits mentioned by Myers (28) in his descriptions of modular software design.

2. CHARACTERISTICS OF PROCESS CONTROL

From the introduction it should be apparent that the two main functions of the process control application program and its enhancements are to control the plant and to supply information about the plant in the form desired. The control function is the primary concern of this thesis and of the chemical engineering department since the methods and results depend on engineering properties such as the chemical process, the layout of the unit and the information available. Also the economic returns gained from this function comprise most of the justification for the cost of the computer installation. The latter function is a data manipulation and storage problem which is more closely related to conventional data processing.

2.1 Structure of the Control Problem

The control problem is broken down into three levels, namely regulatory, supervisory and managerial. The criteria for the division of the control problem between these three levels include

1) Time constant of the process response
2) Complexity of the control algorithm

The relationship between the physical division of the process plant and the division of the control problem is as follows.

"Regulatory control" or the first level is concerned with maintaining a process value (level, flow, temperature). On occasion it may involve two loop cascades. This only occurs on very simple pieces of equipment such as a pump or tank. At this level of control the execution of the control algorithm is very fast (approximately one second) dictating that the algorithm must be fairly simple. There is usually only one value that varies with time in the algorithm, the measured input value. Similarly, the number of parameters that can be varied within the algorithm is also fairly small.

"Supervisory control" or the second level covers the remainder of the online control problems. This group has applications that vary greatly in complexity and execution interval. "Supervisory Control Applications" usually generate the returns that justify the installation of a computer. The fact that each application requires alot of tuning to each individual plant dictates that this area has a large manpower requirement. Applications in this group fall into one of the three categories; online continuous applications, periodic (interrupt driven) applications and data generation (for reports) applications.

"Managerial control" concerns the broadest or plant level of the process. It also is affected by conditions such as the company's policies and the economic environment which are outside of the realm of the plant itself. This level is very complex due to the large number of inputs and unknowns. Control strategies executed in this group depend largely on models which attempt to simplify real situations. Typically, the method used in the calculation of the outputs is linear programming. Due to the size and complexity of the algorithms tackled at this level the period between executions is very long. Often the actual execution is done in a lower priority, background mode so as not to affect the performance of the online applications.

An analysis of the three levels of the control problem illustrates the following. In the procession up the hierarchy the execution interval varies from one to ten seconds at the regulatory level; from minutes to hours at the supervisory level; from daily to weekly intervals at the managerial level. In each execution interval, at all levels, input variables must be read, an execution of the control algorithm occurs and new output values are sent to their destinations. Thus the control algorithm execution duration and the number of inputs and outputs accessed are dictated by the period between executions. It follows that the complexity of the control problem increases in the procession up the hierachy. Since the interval of execution of the control algorithm is so critical it is necessary to examine what determines this interval.

The question of how the period between executions is determined is answered by using sampled data theory. A basic sampled data theorm is that the sampling interval of a signal must be one half the period of the fastest frequency of interest. (Another way of stating this is the sampling frequency must be twice the highest frequency that it is desired to detect.) The indicator of the smallest interval where a change occurs is the time constant of a process. The sampling period is picked to minimize the loss of information and to minimize the quantity of data. The sampling rate usually will be larger if the response is non-linear and very complex. A second fact is that a control algorithm can control only as well as the inputs represent the process information or status. Consequently, theory dictates that any control algorithm has a minimum execution rate beyond which the guarantee of control is lost.

A value of one-tenth of the time constant is used to illustrate the variations in the execution period of the three control levels. Regulatory control of flow or the temperature of two mixing streams has a time constant of seconds. This fixes the execution interval of any control algorithm at one second. A value of one second as a minimum frequency of execution was used because it provided fast enough response for most process control applications while still keeping the load on the computer at a manageable level. At the same control level a tank level or temperature has a time constant of one hour. Therefore in this case the execution can be up to every six minutes. The reason the latter is included in the "regulatory control" level is the complexity of the algorithm is similar to the algorithm on the faster flow control. At the supervisory level where process units and complexes are the targets for control the time constants involved are in the order of hours and thus the execution interval is in the order of minutes. With the managerial level of control the orders of magnitude are days and hours for the time constant and execution intervals respectively.

The rationalisation of the correlation of execution interval and the time constant deteriorates if the process is very non-linear. This also applies for a linear process where a tight quality specifications must be maintained. In such cases it is often economical to attempt to control the dynamics instead of just the overall steady state operation. The span of the frequency spectra of process dynamics, is a number of times greater than the steady state response of the process. Thus in order to examine the dynamics of a process, the sampling rate of the inputs and the execution rate of the control algorithm must be increased proportionately. An additional deterrent to implementing dynamic control is the algorithms become very complex and consequently the execution time restricts their use.

Up to this point reference to an execution interval means the interval is fixed. This is accomplished by scheduling the execution of the control algorithm at a constant interval. This is an important fact since all the control algorithms assume this. Control algorithms are based on simplified models of how the process reacts. The algorithms must be tuned by varying parameters within the algorithm to a particular process to reduce the effect of the imperfections of the model. This tuning process takes into account the amount of information received during each interval or the execution and sampling rate. Accordingly, the tuning parameters would have to be varied if the execution interval varied to maintain a constant control performance. The simple tuning of the parameters for a fixed interval is very difficult and is often reduced to trial and error. As a result a dynamic tuning algorithm would be very difficult (if not impossible) to devise. Such an algorithm would have to be included with the control algorithm if the execution interval varied, which would greatly increase the execution time.

The technique to ensure the consistancy of the execution interval is fairly simple. Typically, the real time feature of the computer software is called upon to schedule the programs at a specific interval relative to a real time clock. Provisions have to be made to halt an execution should the real time feature start to schedule the program again before the previous "scan" is completed. This is essential to maintain a constant interval. Should the execution of a program be terminated due to a timed scheduling of the program, the control algorithm being executed and subsequent algorithms remaining would experience a "time-out". This occurence is indicative of a computer that does not have enough resources for the load attached to it. The time-out feature ensures a steady execution interval for the algorithms that do get processed. It is imperative that the control loops are prioritized in order to ensure the most important ones are always executed.

The one occurance that no amount of software checks and defaults can combat is the failure of the computer hardware. In this instance all control action is halted until the computer can be replaced or returned to service. The only recourse for this type of failure if computer control is to be maintained throughout is to have a backup computer standing by. Rather than having the backup computer simply waiting for a failure it should be performing a useful function. If it is performing process control as well, the matter of switching the work of the failing computer would be simple since all the programs used are already loaded. All that would be required is the transfer of the information pertaining specifically to the applications running in the processor that is failing. This configuration describes a network made up of a number of processors to accomplish the control objective. The fact that all the computers must communicate with one another places an additional burden on the operating system. With this function included in the design specification the network now operates as a type of a distributed computing system.

2.2 Characterization of the Process Control Application

This section on process control applications outlines most of the types of problems experienced. An analysis of these applications will determine the structure of the distributed computer control system.

At the lowest level the primary function is regulation. A simple algorithm, common to several loops, that executes rapidly is required. As a result the data necessary to characterize a particular loop is minimal. The data structure is fixed, simple and repetative. The data required from other processors is likely to be in two forms. The first is a change in the parameters of the algorithm sent from a computer that is in a higher level in the hierarchy. The frequency of such changes is fairly low. The second form is input data coming from processors in the network. The sending processor may be at the same level or at a higher level in the hierarchy. It is hoped that the number of changes of this type is limited since each point adds a considerable communications load. The characteristics desired at the lowest control level are best implemented with a series of autonomous processors. Due to the speed of response necessary the data-base must be stored in core memory. The data-base must be copied to disk and all changes accounted for in order to be recoverable from a processor failure. This assumes that information stored on disk is secure. The placing of the data-base in core avoids the retrieval time from disk.

The second level is much harder to analyse since it encompases a much larger variety of applications. Each application has its own function and accordingly its own data structure, execution time and output characteristic. Communication to receive data from the process or other processors will be in the form similar to the lower level, namely on a demand basis. A problem that arises at this level, since nearly all data has to be requested, is the synchronization of the data. Care must be taken to prevent excess skew or checks must be made to test and react to it. One fact which eases the design problem is that there is little if any sharing of tasks. A program in any one processor will do all the data and mathematical manipulation required rather than delegate such tasks to another processor. This fact reduces the communication and synchronization problem somewhat.

The time interval between executions is typically in the order of minutes. As a result the data-base can be stored on disk since the retrieval time is not significant relative to the execution time of any application. The one demand for fast execution is created if the processor is an interface for communication for a lower level processor. This dictates that the communication's facet of the operating system enhancements must be core resident.

Up to this point all the discussion has been based on applications that run periodically. No mention has been made of cases which are interrupt driven. These cases could be dealt with as a regularly scheduled application testing an execution flag or through use of the hardware interrupt system of the computer. In the first solution, the speed at which action must be taken will determine the testing frequency and the priority of the program that is subsequently scheduled.

Processors operating at the second (supervisory) level in the hierarchy must also run independently. Since the time constraints are not as severe the data-base can be disk resident. Due to the greater demands, the computers must be more powerful in terms of computational speed, complexity of functions and methods of output than those implementing control at the first level.

The third (managerial) level of control could be done off-line in a batch mode or as a low priority "background" job. The process information would be dumped to the computer prior to executing the large managerial type programs. The results would be fed to the on-line processors in a similar manner.

This chapter has characterized the process and the control problem in terms of the chemical engineer and the computer scientist. The remaining portion of the thesis deals with the implementation of the features required to ensure reliability and to provide distribution of the application program.

3. SYSTEM RELIABILITY

The primary objective of control is to maintain the process plant at the the operating conditions at all times. This means that the operating point of the plant must be sufficiently far from the operating limits to accomodate for disturbances which upset the normal operation. The distance the operating point is from the operating limit depends on the expected size of the inevitable disturbance and the rate the control action can counteract the disturbance. Computer control allows the process to be run closer to the operating limits by reducing the size and the effect of the disturbances through faster and more complex control methods. Thus one danger of computer control is the plant is left very close to the operating limits after a computer failure. This could result in loss of material and profits if a disturbance, which accompanies a computer failure, pushes the plant beyond these limits. As a result the system reliability of a computer control system is a very key feature.

The relability of a computer control system is a function of the relability of the individual components and its ability to recover from deviations from normal computer operation. Reliability refers to "the degree to which the system satisfies the expectation that at any point in time the services and resources supported by the system perform their function correctly and are available to the user."[1] The reliability of the hardware is measured either as the number of operations performed correctly before an erroneous result is expected or the mean time between failures (MTBF). It usually improves with the increase in unit cost since it is a reflection of the quality of the component's materials, the severity of the component's testing and the strictness of the component's specifications. The reliability of the software is a reflection of the number of bugs or cases not handled by the programs. It improves with systematic and logical program design and testing.

Recoverability is a far reaching topic because it is imbedded in the normal system operation as well as involving the actual recovery mechanism. The normal operation of the computer must constantly note (log) any significant changes of state. In addition to the logging facility the actual recovery mechanism is made up of two separate functions. These are the identification function that detects the occurance and the severity of a fault and the reconstruction function that either corrects the fault or returns the computer system to the state prior to the faults occurance.

---

[1] pg. 9, "Reliability Issues in Distributed Information Processing Systems",Proceeding of the Annual International Conference on Fault Tolerant Computing, pp. 9 - 16, 1979

3.1 Data Integrity

The first stage of the DISCO program uses a table or database which contains the parameters required by the computer to perform process control. This implementation results with the reliability of the system being dependent upon the integrity of the data in the database. Data integrity refers to the likelihood that any entry will be correct. The data in the table or database may be incorrect as a result of enterring the wrong data initially, changing the existing data erroneously, or erasing the data unintentionally. The inherent integrity checks can only address the last two types of data violations. This stage in the development of the DISCO program attempts to increase the reliability of the program by adding features that prevent erroneous changes and enable the computer to recover from a database erasure.

3.1.1 Erroneous Updates

The DISCO program is presently very susceptible to erroneous updates. The following cases illustrate the various ways this could occur.

1) The database in the DISCO program is placed in an area that is accessible to any program running in the system. As a result any of the programs may use this area as a storage location and unknowingly write over part of the database.

2) A program that intends to make an update has no set

format to check the update is performed correctly. Thus a program may mean to change only three words and actually change six. The additional three words would become corrupted (dirty) data.

3) The final area of concern is the consistancy of correctly executed updates.

This term refers to the maintenance of a database in the form that reflects the actual sequence of events. If care is not taken in the design of the sequencing of events erroneous states will result that do not reflect the actual conditions. This is best illustrated with three examples taken from Grey [2].

(1) Lost Updates

If transaction T1 updates a record previously updated by transaction T2 then undoing T2 will also undo the update of T1. (i.e. if transaction 1 updates record R from 100 to 101 and then transaction T2 updates R from 101 to 151 then backing up T1 will set R back to the original value of 100 losing the update of T2.) This is called a Write / Write dependency

(2) Dirty Read

If transaction T1 updates a record which is read by T2, then if T1 aborts T2 will have read a record which never existed. (i.e. T1 updates R to

---

[2] pg. 431, J.N. Gray, "Notes on Data Base Operating Systems", Lecture Notes in Computer Science, 60, Springer-Verlag, 1978.

100,000,000, T2 reads this value, T1 then aborts and the record returns to the value 100). This is called a Write / Read dependency.

(3) Un-repeatable Read

If transaction T1 reads a record (T1 does not comit) which is then altered and committed by T2 and if T1 re-reads (in the case of a redo of T1) the record then T1 will see two different committed values for the same record." Such a dependency is called a Read / Write dependency.

3.1.2 Erasing of data

This condition usually occurs as a result of a hardware failure. In such a case the volitile memory where the database is located is altered or erased altogether. The only solution to such an occurance is to rebuild the database up to the ocurrance of the failure. This requires a secure (unaffected by the failure) copy of the database at some point in time (checkpoint) and a list of all the changes that occurred between the time of the checkpoint copy and the time of the current failure (system log).

3.2 Comparison of Requirements

This section summarizes the characteristics of a process control system. The characteristics of a real time data processing application (banks, air lines) are outlined. The differences between the applications are discussed with the aim to note how these differences reflect the method of implementation of integrity.

3.2.1 Process Control

A process control application has only a limited number of standard operations. A table or database contains all the information specific to each application. An execution interval is defined as a length of time in which all applications are executed. The processing of each application within any execution interval may only decide that the application is not to be executed. There are stringent requirements on the length of any one execution interval.

The database contains two types of data. One type is seldom changed. However, when an update occurs all integrity measures are necessary. The second type of data has the characteristic of changing every execution. Any values contained in this classification are pertinent to a particular time and do not require elaborate integrity measures. The quantity of data in the first category is larger than the second type.

3.2.2 Data Processing

Data processing applications such as those found in banks and airlines also have a limited number of operations. These operations are performed in a random order. The data bases are very large. Response time for an operation is important but not critical. All operations that change data require full security measures to ensure none are lost.

A comparison of the summaries of the two characteristics reveal a number of differences. The difference which has the greatest effect on the implementation of the executive controlling the execution of the process control application is the requirement that all control activities be executed in a constant execution interval. Such a requirement necessitates the use of all techniques that reduce the system overhead. The reduction of computer overhead is aided by the fact that small databases are involved relative to the data processing applications, the order of execution of the operations is fixed versus the random nature in a data processing application and only some changes to the database need full integrity measures. From this comparison, the changes required to the standard integrity techniques used in data processing applications can be identified and implemented. A discussion of this process follows in the next section.

In both process control and data processing applications each task would require the following three steps:

(1) Read a command string

(2) Perform the specific operation using the parameters in the command string

(3) Generate a response as dictated by the result

of the operation

An operation including all of the three steps will henceforth be classified as a transaction. The advantages of defining a transaction are twofold. First, it formalizes when an operation is complete and consequently must be duplicated when rebuilding a database. Secondly, it defines the operations the interface program (database manager) has to perform. Further characteristics of a transaction will be brought out in the following section.

3.3 Transactional Analysis

A transaction is any process that has a beginning and an end whose effect is not realized by the rest of the system until the transaction is completed. The last operation of every transaction, a commit, is the passing of the effect onto the system. The defintion of commit is imperative for a recovery mechanism because it eliminates the indecisiveness where an operation can be partially complete. Related terms which are associated with a transaction are:

Transaction descriptor:

a record in the table driving the transaction processor which defines the operation to be performed, the values of the parameters to be used and the action to be taken as a result of the result of the execution of the transaction.

Process:

the set of instructions that direct the manipulations to be performed in the execution of the transaction. It includes the scheduling of the necessary resources. It can work on only one transaction at a time.

Transaction environment:

all the resources and processes needed by a transaction.

Resource

any part of the system that can handle only one operation at an instant yet must be used by a number of processes. Examples of these are devices such as discs, printers or card readers, processors such as array processors or arithmetic units and files.

The first description of the concept of a transaction was detailed by Davies (15) and Bjork (4). In these papers the transaction was characterized by its Sphere of Control. This term refers to the bounds in which the transaction operates and is defined to control the resource commitment. The primary object of the paper was the logical definition of the sequence of a process paying particular attention to the requirements for integrity. This leads to detailed discussion of the use of resources and the dependencies created between transactions through the sharing of common resources.

A secure transaction is defined by a procedure involving four steps. This procedure is only required by operations changing data located on resources which store information.

(1) Initialize Transaction

This step involves the interpretation of the transaction descriptor, the testing of the validity of it and the attaining of the required resources. The resources would be held until the transaction is complete.

(2) Creating of the Transaction Log

This step establishes a record to be entered in the transaction log that enables a transaction to be obliterated if it fails. The recovery method is such that the system will not know of an incomplete transaction's existence. A string that contains the old value of the database is not required since the database is not actually updated until the commit phase is entered. The use of this record is to repeat an operation if it encounters a mild error such as a time-out, which precludes normal termination or to rebuild the database after a failure.

One feature of any transaction is the ability to be repeated without changing the net result. This quality prohibits operations which increment existing values since the final value of a data item would vary depending upon the number of times the transaction was repeated.

A typical record in the transaction log would contain the transaction identifier (a value which makes each log unique), a transaction status flag, the time of execution and the command to change data contained in the resource. A record of this form provides the information to return to the old state as well as retry the same command for mild errors.

(3) Execute Transaction

During this step the actual changes to the resource are calculated. This is the step where the process is executed, the results are found and placed in a buffer awaiting the commit command to actually alter the data stored in the resource.

If errors are found then the transaction is "aborted" and all resources are returned to their old state using the data contained in the transaction log. A successful transaction then passes into the next step.

(4) Commit

This is the end of the transaction where the change caused by its execution is written to the database and passed onto the system. The first step involved in the "commit" stage is the setting of the transaction status flag in the transaction log. Finally, the successful change is entered in the database. This last step finalizes the transaction since it can no longer be undone.

A transaction as defined simplifies the recovery of a computer since it reduces all operations that require time for processing to events that happen at a specific instant, namely, at the end of the operation. A backup copy of the database or "checkpoint" can be taken whenever there are no transactions locking any part of the database. A rebuild of the database after a failure simply requires the reloading of the checkpoint copy and updating it from the log of commited transactions (audit trail). Any transaction that was in progress is ignored since the system is not affected by them until they are committed.

Transactions relate specifically to the process control application. An activity which constitutes one control problem is a transaction since it begins by reading data, processes the data, then changes the system according to the results of the operation. The characteristics of consistancy and recoverability of the process control application can now be analysed.

3.3.1 Locks

The first action a transaction makes after the translation of the transaction descriptor is the identification and aquisition of the resources required to complete the transaction. The first step in accomplishing this action is the checking to determine if the resource is in use. If it is the transaction is queued to be restarted when the resource is free. Once the resource is acquired by a transaction it is locked to notify any other transaction that the particular resource is busy. One problem with this mode of operation is that one transaction can lock a number of resources and thus delay a number of other transactions that require the same resources. This can escalate into a complete system stoppage.

3.3.1.1 Granularity of Locks

The degree a resource is locked depends upon the resource and the action being performed on it. The resources that do not store material are easily accomodated since they take the input stream and process it. Locks are not necessary since the device is busy or it is available.

This leaves the storage devices such as files and discs which are more complex since the information that they contain is time dependent. The question that must be answered is when and what type of locks are required for storage devices.

The two processes that can be performed on a storage device are a "read" and a "write". In the "read" process there is no change in the database during execution. This means the order of successive reads is immaterial since they all will return the same data. As a result it is not necessary to lock a database during a "read" process.

The "write" process is different since the database changes after the "write" command is committed. Thus the time a "write" occurs is important since it signifies a change of state in the database. It is necessary to lock a database or file when performing a "write". No uncertainty of the value obtained is possible since any attempt to "read" during a "write" would be rejected by the presence of the lock.

The next problem is the instance when a "read" is being performed and a "write" begins. With the protocol outlined above the "write" process has no knowledge of the "read". Thus the "write" will lock the file and start to execute. The value the "read" attains now depends on the extent the "write" is completed. There must be a feature to prevent this occurrance. A solution is to have a lock with levels of severity. The lock of a resource which is available would have the lowest level or 0 signifying it is unlocked. The lock of a resource in the "read" mode would have the value of 1 signifying it is a mutually sharable mode. Finally, the value of a lock in the "write" state would be 2 signifying an exclusive mode. The exclusive mode can only be obtained if the lock is initially unlocked.

One area not discussed is the extent of a lock. The need for this is best described with an example.
If a "write" is being made updating a word in a record in a database, the lock may hold the whole database, the particular record or the paricular word which is actually

being changed. The decision depends upon the application and how it uses the database, the records and the words. An additional factor is the size of the system which dictates the number of locks which are supported.

3.3.1.2 Consistency

The one method to ensure consistancy is to lock all the resources required for the transaction. This guarantees that the resource will see only one transaction at a time. The problem with this is the computer system essentially becomes a serial processor since only one transaction of a group sharing resources can be run at one time. This greatly reduces the systems efficiency since it is held in a wait state while the slower devices complete their tasks.

A relaxation of this scheme is to sequence the transactions in such a manner that the model of a serial processor is preserved. There has been extensive work done to develop better scheduling algorithms to accomplish this. Papers by Bernstein et al (3), Stonebraker (39) and Thomas (40) describe some of the methods used.

3.3.1.3 Deadlocks

Deadlock is the condition where two programs which are running simultaneously cannot acquire all the resources necessary to complete a transaction. This is because each has a subset of the other program's resources locked. The result is both transactions wait indefinitely for the desired resources to become available. An example of the phenomenom follows:

> Program "A" and "B" both require the card reader and printer. Program "A" initially locks the printer successfully because it is available but cannot lock the card reader because Program "B" has it locked. Next Program "B" finishes using the card reader but does not release it since it is needed again. Program "B" tries to lock the printer but cannot since Program "A" owns it. The result is both Programs "A" and "B" stall with the card reader and the printer unavailable to the system.

The solution to this problem requires either a deadlock prevention system or detection system. A deadlock prevention system will be discussed first with the detection and resolution systems following.

A method of deadlock prevention is the two step procedure that requests, locks and runs only when all resources are available. This mechanism is simpler than the priority system but the utilization of the resources deteriorates since a program will lock all resources for the complete execution duration while only using them for a portion of the time. This scheme also causes delays in the execution of the transactions since the resources will be locked out when the transaction processor makes the resource request.

An alternate approach to deadlock is to assume it will occur and have mechanisms that detect and resolve the contention problem. The detection scheme usually involves some form of graph analysis for loops or cycles. The complexity of the detection mechanism is proportional to the complexity and size of the application. The one drawback of detection mechanisms is the need for the task to be monitored by a single processor. Deadlock detection cannot be accomplished by a number of separate autonomous units because they do not have sufficient information of the rest of the system. Thus the detection mechanism is centralized and must be backed up.

Once a deadlock is detected it is resolved by holding one program which then releases its resources. The decision as to which program is held may be random or may involve a priority scheme. Such a scheme may cancel the consistancy property that the locking mechanism ensured. However, this is necessary to return the computer to full operability.

The decision of which scheme to use revolves around the common paradox of efficiency versus overhead that enters into most computer design problems. The decision is also largely application dependent. For example, the two step prevention scheme works fairly well if the application has short transactions and the resources are lightly loaded. In such a case any resource is held for only a small duration so the probability that another transaction would begin and want the same resource is small. A process control application is such a case because the computer has a number of short fast transactions that use one type of resource. Also most of the CPU cycles are tied up with longer slow programs that are using separate resources.

3.4 Recovery

The recovery mechanism is the set of steps that must be taken to correct a minor fault or to return the system to the state prior to the occurance of a major fault. This section outlines the needs of such a mechanism and the various options available to accomodate these needs.

Recoverability is composed of two separate functions. These are the identification function which detects an abnormal condition and the actual recovery function which returns the system to a normal operating mode. The identification function is extremely complex since it must operate while the system is crashing down around it. There are a number of levels of failures which the identification must recognize and signal.

(1) Operation failure: where a particular operation used in a transaction does not work due to bad parameters, exceeded memory requirements or insufficient data, etc.

(2) Transaction failure: where a series of operations grouped together as a transaction do not proceed due to time outs, deadlocks, protection violation or resource unavailable,

etc.

(3) System failure: where a subprogram within the operating system fails due to mapping violations or wild branches, etc.

(4) Resource failure: where a piece of hardware fails. Such a failure is particularly serious if the resource is the non-volatile storage media (disk) since all the copies of the system and application programs are not available.

It is the last two groupings that are the most serious and as a result have the greatest effect on performance. With errors in the system and resource failure categories all programs running will be effected and thus the system stalls very quickly. Such failures also allow the least amount of time to take corrective action.

The actual recovery aspect attempts to return the system to the normal operating mode. This usually requires two steps. First, the offending program or resource must be deactivated. This step might include the activation of a redundant, stand-by resource to fill the gap left by the ailing part. The second step returns the system back to the state just prior to the fault that caused the system outage.

The steps involved in the reconstruction of the system state depend upon the type of resource that failed. Non-storage devices will simply be restarted. The steps required for storage devices vary depending upon the method of backup. There are two basic methods although many variations and modifications of the basic method are being used.

The first method, called Dump/Restore, backs up a data base by taking a complete copy of the database (checkpoint) and storing it on a non-volatile (disk) memory device. In addition, a record of all the changes to the database, since the last secure copy was made, are kept in the transaction log (audit trail). These may be in the form of change commands or may be a copy of the changed record. When the database needs rebuilding the secure copy is read in and all the changes logged in the audit trail are appended to arrive at an up to date version.

A "checkpoint" or complete copy of the database is taken between transactions. At this time the transaction log is reinitialized to be restarted with any changes that subsequently occur. The record in the transaction log of the successful changes make up the audit trail. The interval between checkpoints is determined by analysing the loading on the system and the speed of recovery desired. Increasing the interval between checkpoints lowers the overhead of the copying of data required in the checkpointing operation. The problem is the audit trail grows to a substantial length near the end of the interval, since it contains all the changes made from the last checkpoint. If a fault occurs then it would take a considerable time to run through the complete audit trail to rebuild the database. If the checkpoint interval is small the audit trail stays fairly short but the overhead of the continuous copying of the data base gets substantial. Chandy (8) has done some work on modelling systems and applications to optimize the checkpointing interval.

The second method, called Duplication, backs up a data base by maintaining a copy on a separate storage device (eg: the database in core memory and a copy on disk). All changes alter both copies of the database. The database is reconstructed by copying the duplicate back into memory. This is a very fast mechanism. The method requires special protocols to ensure both copies are indentical at all times. This procedure is slower than the Dump/Restore when operating in the real time mode since all change commands must be executed twice. The other drawback is there is no record of the various changes that have occurred to the data base.

The Dump/Restore method of backup is fast during the normal packing operation but is slow in the reconstruction since the complete audit trail must be reprocessed. The Duplication method is slower during the normal integrity operation but is fast in recovering since the backup copy is simply reloaded. The Dump/Restore method is used in process control applications since the operating overhead is most critical and needs to be minimized whereas the length of the restore operation is less critical.

3.5 Application Integrity

The preceeding discussion of integrity has focused on the concept of a transaction with emphasis on databases. Application integrity refers to the features that ensure the consistancy of the execution environment of the various control activities which make up the overall process control application. The consistancy is necessary to guarantee the predictability of the control action. The type of action which destroys the consistancy is the partial execution of a control activity such that only some of the output variables are updated. This situation results in an inconsistent state which is defined as any condition where the values in a single control activity have been updated or renewed at different times. The solution to this problem is to design the control activity to function as a transaction. This would ensure that the activity cannot be partially complete.

The activity that falls into the classification of a transaction uses the following procedure. The activity record would be read, translated and interpreted. The resources necessary to perform the activity would be attained. The parameters translated from the activity record would be used in the process also specified in the activity record to calculate and buffer the results. When all the processing is completed and all the results are available they would be passed onto the system and hence be available to any other activities. This procedure guarantees time consistancy of the control activities.

In the event of a failure, the control application that was being processed would simply reread the input data and start processing again. These two steps are all that is required to restart after a failure.

3.5.1 Restore versus Restart

A restore is a procedure that is executed after a major failure where the databases are reconstructed from the checkpoint and audit trail and the system starts operating using the old values. In a process control application this can only be done if the duration of the failure is small, since the values contained in the database are time dependent. The alternate to restore is a restart where only the data that is not time specific is reconstructed from the checkpoints and audit trails. Normal operation proceeds only after all the time specific data is obtained by resampling the process. The restart procedure can always be executed but it takes more time to return to normal operation.

To conclude, a program called ACCES was written which isolates the database from the other functioning programs. It is the interface program and creates a format which must be adhered to read from/write to the database. The requests are treated as transactions to guarantee the consistancy of the database and to provide a log of the changes as they occur. The log is used to regenerate the database.

The actual design of the ACCES is discussed in chapter

5. The testing and test results are contained and discussed in chapter 6. The next chapter deals with the second topic of the thesis, Distributed Process Control.

4. DISTRIBUTED SYSTEMS

The first application of computers for process control had the process inputs and outputs connected directly to the single computer. This architecture limited the scope of the problems that were possible since the plant would still have to operate in the event of a computer failure. The second stage of development added an additional computer standing by to take over in the event of a failure. This broadened the scope of applications possible but was inefficient and expensive since one computer was always idle. The last step in the evolution uses a number of computers to share the task of the system, yet can change the specific task of any processor depending upon the state of the system. In this thesis "distributed system" refers to any system where the task is split amoung a number of semi-autonomous computer systems. The load and function of any processor varies depending upon the state of the system.

4.1 Communications Requirements

The objective of this portion of the thesis work is to develop a communication package that fulfills the efficiency and reliability requirements of a computer network used for process control. Prior to designing and implementing the program it is necessary to determine the communication needs of a distributed process control system.

4.1.1 Process Requirements

The transmission rate requirement varies depending upon the division of the control tasks. The message integrity requirement varies for the type of information sent. Communications which are frequent and contain only time specific values such as process data do not need a sophisticated security scheme since the data will be updated in the next communication. The other extreme is a communication that needs full security measures since it contains unique updates to control parameters that will be sent only once.

4.1.2 Process Applications in a Multiprocessor Environment

The communication link between processors used in the design of enhancements for DISCO was relatively slow. This severly restricted the communication. Thus any data transmitted would not be updated frequently. The result is that each processor in the network is fairly autonomous using mainly local data. However, there are cases where two processors are necessary to accomplish a single control task. These typically fall into three categories.

4.1.2.1 Cascade Control

The set of applications which are grouped in this category involve a control strategy that requires two execution intervals. Two processors are required if the process values are separated geographically. A simple case is a level cascaded with a flow where the level is measured in a reactor in one complex and the flow is in a different unit. The separation of the application over two processors is possible only if the ratio between the two execution intervals is greater than three.

4.1.2.2 Signal Conditioning

The distributed applications in this group are characterized by input signals that require extensive processing before they are in a usable form. In such cases the processor actually connected to the process acts as an input preprocessor while the second processor actually performs the control function. Typical examples of these applications are those using Gas Chromatograph measurements, those requiring temperature and pressure compensation of flow and those which derive process information from a nonlinear transducer.

This group of applications opens the discussion of distribution by function or by geographic or process location. If the first criteria is met all G.C.s of a plant would be plugged into one processor dedicated to that service. Such a solution requires extensive wiring to bring all the signals to one location. The efficiency of such a processor would be optimal since it was designed for one specific purpose. Now the problem is that the computer is unique in the network and has no backup if it fails. The alternate is to locate each G.C. in the closest processor.

In this instance each processor must contain a complex progam to handle the data. This duplication of function will dictate that each processor must have a larger capacity and a higher cost.

The solution is a compromise with a number but not all of the special inputs handled by a dedicated processor. This solution would dictate that a number of the dedicated processors would be required. The processors would be placed to minimize the wiring to connect the inputs whereas still provide a back-up capability should one fail.

4.1.2.3 Reporting

This is one of the key functions of the computer system and consequently is a concern of every processor in the network. The hierarchial structure aids the implementation since the higher level controls its subordinates in the execution of this function. The network communication function must allow the information maintained at each of the processors to be accessible throughout. Post-processing programs in the processor requesting the information would format the data in the form desired for the report.

This section acts as an introduction to the subject of processor loading and location of programs and databases. As expected, the answers to such questions are almost totally application dependent. The distribution of the process points and the type and complexity of the application programs are the factors that affect the distribution. Baker (2) illustrates one such analysis for a large steel plant. It is only through the type of analysis described in his paper that a system can be optimally utilized.

4.1.3 Comparisom of Process Control with Data Processing

The communication requirements of a process control application are similar to those of a data processing application. The requirements fall into two categories, hardware and software. The hardware level is concerned with the actual specification of the communication link (bandwidth) and the reliability (message sent is the message received). The software level is concerned with the reliability and recoverability of the communication messages. An example of the latter is when one processor sends a command to three other processors in the network. If only two of the recipients receive the command the co-ordination between the three processors is lost. A process control example of this problem is the case where two units drawing steam are instructed to increase their demand but the utility plant does not receive the command to increase steam production. In any computer application, whether process control or data processing, it is desirable to ensure that all destinations receive a message.

In a process control application the assurance of this characteristic is important but not critical because there are other separate mechanisms that illuminate the error and correct for it. In the last example the fact that the utility plant did not receive the command would become obvious because the steam demand would increase causing the production to increase through a separate control loop. The result would be a bump in the steam quality but the data base would be corrected.

In a data processing communication the alternate checking mechanism is often not present or very slow. In both cases the application operates on incorrect data for a considerable time making corrections very difficult. As a result these applications require a higher degree of security to guarantee that all communications are complete.

4.2 Communication Program

The protocol for a distributed process control system should have a minimal effect on the performance of the communication media yet have enough capability to guarantee messages sent to one or more processors are received. The work in this area first, analysed the features of the existing vendor's communication package then added features that increased functions of the existing package to the requirements of the distributed process control system.

4.2.1 H.P. Communications Package

The protocol used by Hewlett Packard is as follows:

(1) The master sends a request that it wants to communicate.

(2) The slave replies it is ready.

(3) The master communicates the message.

(4) The slave replies the communication is successful/failed.

The message lengths are variable and the error checking mechanism uses a two coordinate parity code.

This protocol handled the problem of communication between two processors. If any errors occurred, both processors were notified and the necessary action could be taken. The shortfall of this protocol comes when one processor sends a message to a number of other processors in the network. In the event of a failure only some processors would receive and act upon the message. The processors which did not receive the message would not know of its existance. This undetermined state is not acceptable. Thus an addition to the H.P. communication package is required.

4.2.2 Additions to the Communication Package

The addition to the H.P. communication protocol uses the same method of Initiator/Send and Target/Acknowledge as the Hewlett Packard protocol. The addition is a polling mechanism that ensures all the receiving processors have completed each action prior to proceeding to the next step. This polling procedure can be likened to a Synchronization step. The actual Send step occurs after all the computers involved in the transmission have been contacted.

The protocol that was implemented is outlined in Figure

4.1. The communication procedure is very similar to the procedure used for a secure write to a database as described in chapter 3. The comparisom is made in Figure 4.2. This fact means that the communications procedure constitutes a transaction (labelled "Send"). DISCO working in a distributed environment is still a transaction processor. As a result distributed DISCO has all the integrity and reliability characteristics of the version that operates in one processor.

An unknown condition may occur in step 3. Up to this step the "send" transaction can be undone in the remaining nodes should one fail. However, if a slave or receiving node fails during step 3, the system log of the failing computer has an entry which is incomplete. On a rebuild of the data base the computer does not know if it should "undo" or "commit" since this information was sent during the failure. This dilemma is solved by checking the system log and audit trail of the master or sending computer for the message number in question. If it is found the transaction was successful, then the computer will "commit". Otherwise the computer "undoes" the transaction. This check lengthens the return of a computer to normal service but ensures that the database is correct.

From the discussion of distributed computer systems it is apparent that they can meet the requirements of a process control application, as outlined in chapter 2. The areas

	Master		Slave
STEP	ACTION	STEP	ACTION
1)	- write message of communication in log		
2)	- send message to slave computers		
		A)	- receive message - write entry in transaction log - ackn. completion
3)	- wait for all slaves to ackn. - commit communication - send commit command		
		B)	- commit Send transaction - (optional) ackn. completion
4)	- (optional) confirm commit		

Figure 4.1 Protocol for Secure Communication

Secure Write	Network Communication
ACCES - entered	COM - entered
- entry of action in log	- entry of action in log
- write data	- sends commun. string
	- receives ack. of messages
- commits trans.	- commits trans.

Figure 4.2 Comparisom of Secure Write and Send Transaction

where the distributed system excels are those of key importance to the process control application, namely reliability and flexibility.

The remainder of the thesis deals specifically with the design, implementation and evaluation of enhancements of the DISCO program.

5. DESIGN OF SYSTEM

This thesis is concerned primarily with the enhancements to DISCO that permit distributed computing and ensure the integrity and recoverability of the database. In performing the design and implementation of these features, attention must also be paid to the other interacting areas of the program. This in essence dictates that a skeletal design of the complete system is necessary to ensure any protocols are receptive to any functions yet to be implemented. Once this is complete, detailed design is performed and implemented on the two aspects mentioned.

The design of the enhancements will follow the methods laid out in the books by Gunther (20) and Myers (28) on software design. These books stress structured design. This is accomplished by breaking the initial problem into a number of simpler problems. There are two basic steps involved in the realization of the final design. The first is the actual decomposition of the problem into modules. The second step is the analysis of the relationships between modules. The actual design process iterates over these two steps until a simple, easily implementable design results.

5.1 Module Description

The overall object of the DISCO program is to provide functions that facilitate the use of the computer in performing process control in an university and industrial environment. The inclusion of university as well as industrial environment means that a large emphasis is placed on the flexibility of the system. The university environment requires a tool that allows for the implementation of new and somewhat radical control methods. In addition, the system must be able to be tied to another operating program that simulates the reactions of a typical plant. This feature is necessary for the execution of simulation studies prior to testing the control schemes on the actual equipment. The equipment being controlled is an order of magnitude smaller than an industrial plant and consequently has time constants and sampling intervals in proportion. This eliminates any advantages gained from the fact that the number of loops is less than an actual plant because each loop must be executed at a higher frequency. Another difference between the two environments is the importance of the process operator function. In the industrial environment this is one of the key features since the volume of data that must be monitored is extremely large. As a result a great deal of effort is placed in the efficient presentation. The university environment has equipment with only a few critical measurements that must be monitored.

The final layout of the modules and levels is shown in Figure 5.1. The functions of these modules and the time of their development (development "stage") is in Table 5.1. The three stages of development are, first, the initial DISCO program by Andrew Brenek (6), second, the enhancements on integrity and distributed processing, the work of this thesis and third, any future modifications.

5.2 Flow of Process Control Application in Disco

All process control problems can be described as a transaction. Data is gathered, it is manipulated according to a set of rules using a set of parameters and then the result of the calculation is outputed. Disco will be described in these terms.

The first step is the loading of the Activity Record that controls the activity execution. This is necessary as a first step because one of the parameters gathered is the location of the process data. This step corresponds to the EXECU module requesting the ACCES module for an Activity Record. The TRANS module is called to convert the Activity Record received into a meaningful set of parameters.

The second step is the actual activity execution in the PROCS module. Once the rules and parameters are received, PROCS starts the procedure by requesting the input data. For control applications, this is executed by sending a request for data to ACCES through FORM. Next, PROCS calls the CONTL module to perform the control calculations. The results are returned to the PROCS module. The final step is the storing of the results also through FORM and ACCES in the ART. The application is now completed so the EXEC module requests the next Activity Record. This paragraph describes the execution of a control problem with the process input and output data

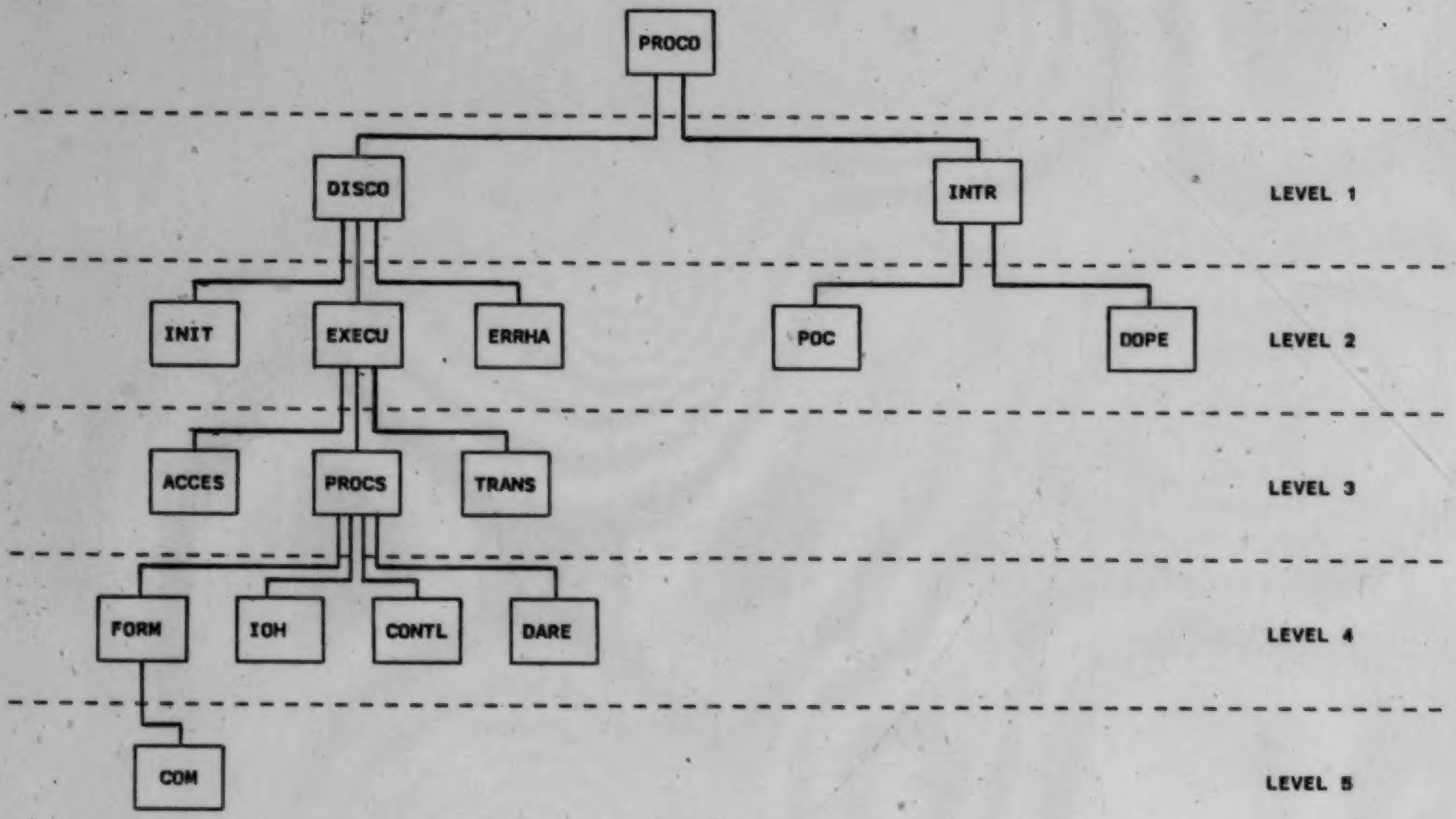
PROCD
DISCO
INTR
LEVEL 1
INIT
EXECU
ERRHA
POC
DOPE
LEVEL 2
ACCES
PROCS
TRANS
LEVEL 3
FORM
IOH
CONTL
DARE
LEVEL 4
COM
LEVEL 5

Figure 5.1 MODULE LAYOUT

Name	Stage	Description
PROCO		- PROcess COntrol application program
	Real Time Portion	
DISCO	1	- DIstributed Supervisory and COntrol executive of all functions
INIT	1,2,3	- initialization of modules; to be expanded to include restarts
EXECU	1	- director of EXECUtion of activities
ERRHA	1,2,3	- ERRor HAndling program; processes error message from the other modules
ACCES	2	- controls ACCESs to the data base; responsible for data base integrity
TRANS	1	- TRANSlator of activity header
PROCS	1,2	- directs the sequential PROCeSsing of the segments which make up an activity
FORM	2	- separates and FORMates local and network data requests
IOH	2,3	- process I/O Handler
CONTL	1,2	- process of the numerous CONTroL segments
DARE	3	- DAta gathering for REport generation
COM	2	- network COMmunication module
	Interrupt Driven Modules	
INTR	3	- overall executive of INTeRmitten ly used modules
POC	3	- Process Operating (Console) module for monitoring of process values
DOPE	3	- builder program for construction and maintenace of data base

Table 5.1 Module Functions and Development Stage

located in the ART.

There is a different routine for the input and output of data to the process. In such a case the first step is identical. The second step varys only in the location it gets the data from process inputs or sends the data to process outputs. In the case of a process input, the PROCS module calls the IOH module for the value of the data. The result from IOH are sent to the ART via FORM and ACCES.

The procedure is clarified further with the flow diagrams in Figures 5.2, 5.3 and 5.4 that detail the process sampling procedure, the control calculation procedure and the update procedure of the calculated variable respectively.

5.3 Detailed Design

This thesis is a continuation of the development of DISCO, an application package that provides a computer environment suitable for process control. The work by Brenek (6), the start of this project, completed the design of the modules that allowed for real time operation. The modules constructed were DISCO, INIT (skeleton), EXECU, ACCES (skeleton), TRANS, PROCS, CALC and a number of sub-modules of CALC. These modules allowed for continuous control on one CPU using an Activity Record Table that was accessible to any program.

This product of the work of this thesis adds to the existing program, functions, that ensure the integrity of

STEP	MODULE	FUNCTION PERFORMED
1)	EXEC	request activity (contains process I/O)
2)	ACCES	retrieve I/O sampling activity
3)	TRANS	convert parameters
4)	PROCS	pass parameters to IOH
5)	IOH	sample the process
6)	CALC	perform conversion of raw value to actual value and limit check
7)	PROCS	format the return string
8)	ACCES	enter updated process value into ART

Figure 5.2 Sequence of Input Segment

STEP	MODULE	FUNCTION PERFORMED
1)	EXEC	- request activity (contains PID control)
2)	ACCES	- retrieve PID control activity
3)	TRANS	- convert parameters
4)	PROCS	- decide processing sequence - pass parameters to FORM
5)	FORM	- decide whether local or network request for data - send request for data
6)	ACCES	- retrieve process data from I/O table
7)	PROCS	- pass parameters to CALC
8)	CALC	- calculate new value for control variable
9)	PROCS	- format returned value for FORM
10)	FORM	- decide whether local or net. - pass string
11)	ACCES	- write new value of control variable into I/O table

Figure 5.3 Sequence of Control Segment

STEP	MODULE	FUNCTION PERFORMED
1)	EXEC	- request activity (contains I/O output)
2)	ACCES	- retrieve I/O output activity
3)	TRANS	- convert parameters
4)	PROCS	- decide processing sequence - pass parameters to FORM
5)	FORM	- decide whether local or network request for data - send request for data
6)	ACCES	- retrieve process data from I/O table
7)	PROCS	- format string for CALC
8)	CALC	-calculate new raw value for output to process - perform limit checks
9)	IOH	- write data to the process

Figure 5.4 Sequence of Output Segment

the database and extend the operation to a network of processors. This was accomplished by redesigning the ACCES module and adding the COM module. Other features added are the ERRHA and an extensive reconstruction of the INIT module.

5.3.1 ACCES

The purpose of the ACCES program is to be the interface between the database which contains the data and programs that perform process control. The isolation of the database, with only one program that can acces it, greatly simplifies the data integrity and database recovery.

The ACCES program handles any request to the database as a transaction. There are three functions; Read, Secure Write, and Simple Write. The "Simple Write" function writes to the database but does not log the updates. This was included to reduce the overhead and is used on data that is updated regularly (every execution interval). The execution of the "Read" and "Secure Write" are handled in the manner described in Chapter 3 (page 34) under the discussion of transactions.

This implementation improves the integrity by addressing both the problem of erroneous updates and of database erasure. Erroneous updates by external programs are now prevented since all changes must go through the interface program. All changes contain the number of words to be alter as well as the new values. This ensures that the update alters only the specified number of words. The design of the program is such to allow for the inclusion of restricted access to areas of the database. This was not implemented because of the additional overhead involved. Finally, treating any access to the database as a transaction ensures that the database always remains consistent with time.

The event of a database erasure is handled by ensuring the database can always be regenerated to the present state. This was only possible by implementing a checkpointing mechanism of the database and a logging mechanism of changes occurring between the checkpoint copy and the present time. Both the checkpoint copy and the log are on non-volitile memory that is unaffected by a crash of the system. The two operations make up the recovery mechanism. The "Simple Write" transaction was required because the overhead of logging regular updates to non-volitile memory seriously detracts from the performance of the system. The recovery mechanism of the database counters the event of a database erasure or massive erroneous change usually occurring in the event of a system failure.

5.3.1.1 Modules

From the discussion of data integrity the modular structure takes the form of Figure 5.5. The logic of the program is illustrated in Figure 5.6. The following discussion identifies the function and the communication

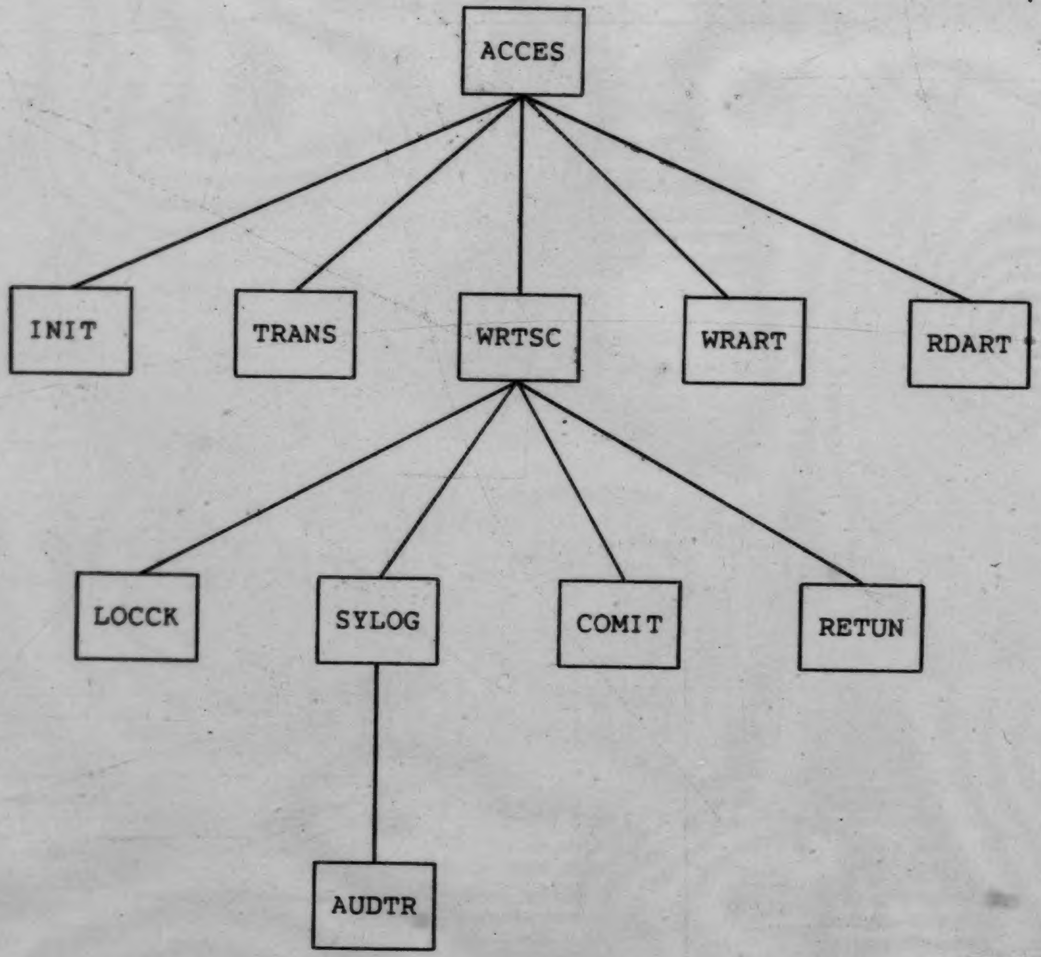
ACCES
INIT
TRANS
WRTSC
WRART
RDART
LOCCK
SYLOG
COMIT
RETUN
AUDTR

Figure 5.5 Detailed Module Layout for ACCES

```
(1)  Read next request

(2)  IF -initialization is required
           THEN -call INIT
           -return to (1)
     ELSE -continue

(3)  IF -translation from Act., Seg., &
           Word to Absolute addressing is required
           THEN -call TRANS
     ELSE -continue
(4)  IF -the request is a read
           THEN -perform the read
           -return the values
           -return to (1)
     ELSE -continue
(5)  IF -the request is for a secure write
           THEN -schedule WRTSC (secure write pgm.)
           -return to (1)
     ELSE -continue

(6)  Perform simple write
(7)  Return to (1)
```

<u>Figure</u> <u>5.6</u> Logic for ACCES Module

string passed between each module.

ACCES

ACCES contains the logic that controls the manipulation of the communication string and manages the Activity Record Table. The first function implemented directly by this module is the stacking of the requests. This is necessary to prevent the loss of a request while an earlier one is being processed. This function is accomplished using a HP operating system feature called Class I/O. A sub-module providing the message stacking function would be required in an alternate system.

The module takes a communication string off the stack and begins processing. The logic must decide the number and the order of execution of the sub-modules necessary to complete a request. Finally, the module formats the communication strings and passes it on to the various sub-modules.

INIT

This module secures the resources for the locking function and the message stacking function and secures the program in memory. The program requires only a status flag as an input and returns the necessary locks and stacks along with any error codes.

This module will need to be extended as the development of the recovery mechanism proceeds since it will contain the modules that return the system to the last operating state prior to a fault.

TRANS

TRANS converts the data from address by Activity, Segment and Word number used by the INDR section to direct addressing (the word number relative to the start of the Acitivity Record Table). The input is the communication string and the output is the modified communication string.

RDART

RDART performs one of the primary functions of ACCES. It takes the command string, checks that none of the requested data is locked, reads the values then returns them to the requesting program. The checking of locks which is performed by one of the two sub-modules of READ ensures the integrity of the data. The formating and sending of the return string is the function of the second sub-module. The only return value to DISCO is the error code.

WRART

WRART performs the second major function of ACCES. It takes the command string, checks that none of the data is locked, and writes the changes contained in the command string to the database. A single word reply is returned upon completion of the change. The objective of the module is the fast update of the database. This module is used for changes of data which are time specific and frequently updated. Another module, described next, controls the updates which require full security checks.

WRITE

WRITE performs the second primary function of ACCES. It is the most complex module since it must deal with both integrity and recovery. These two features are closely related since a database that can recover from a failure has high integrity.

The WRITE module receives the data string from ACCES. It must initially check any locks to ensure the changes requested can be made. An available region of the database is locked to ensure no other accesses occur during the update process. The LOCCK sub-module accomplishes this. The next step is to write to the transaction log. The SYLOG sub-module formats the transaction log record and writes it to disc (non-volatile). The RETUN sub-module sends an acknowledge of the update to the requesting program. WRITE must now wait until the initiating program echoes this acknowledgement before it can finish the transaction. With the echo WRITE activates the COMIT sub-module that completes the entry in the system log, changes the database and frees all the locks, finishing the transaction. The only return from the WRITE module to ACCES is the error code.

5.3.2 COM

The function of this module is to compliment the existing H.P. communication package to provide secure communication amongst the processors making up the process control system. The communications protocol uses a conversation scheme where receipt of messages are acknowledged by the target processor. This protocol, as outlined in chapter 4, also ensures all messages are time consistent. This property is inherent in all operations that can be classified as a transaction.

The COM module is built around the assumption that all communication is performed by sending data in response to a request. If this scheme is not convenient then it would be up to the application engineer to design separate programs that use the vendor's network communication package separately. Such a solution must be analysed carefully since it may impact on the standard communication mechanism.

5.3.2.1 COM Modules

The detailed design results in the final structure of the COM program shown in Figure 5.7. Figure 5.8 and Figure 5.9 are schematics of the logic of the COM module.

The functional objectives require that a COM module can receive two types of requests. The first comes from within the node of the module. This type of request asks for messages to be sent. The COM module must have the capability to collect any responses to the message sent and return them to the initiating program. The second type of request comes from a COM module in another node. The COM's function is to pass the request onto a program that can fulfil it.

COM

COM is responsible for stacking the requests if the program is busy, initializing the transaction and sending

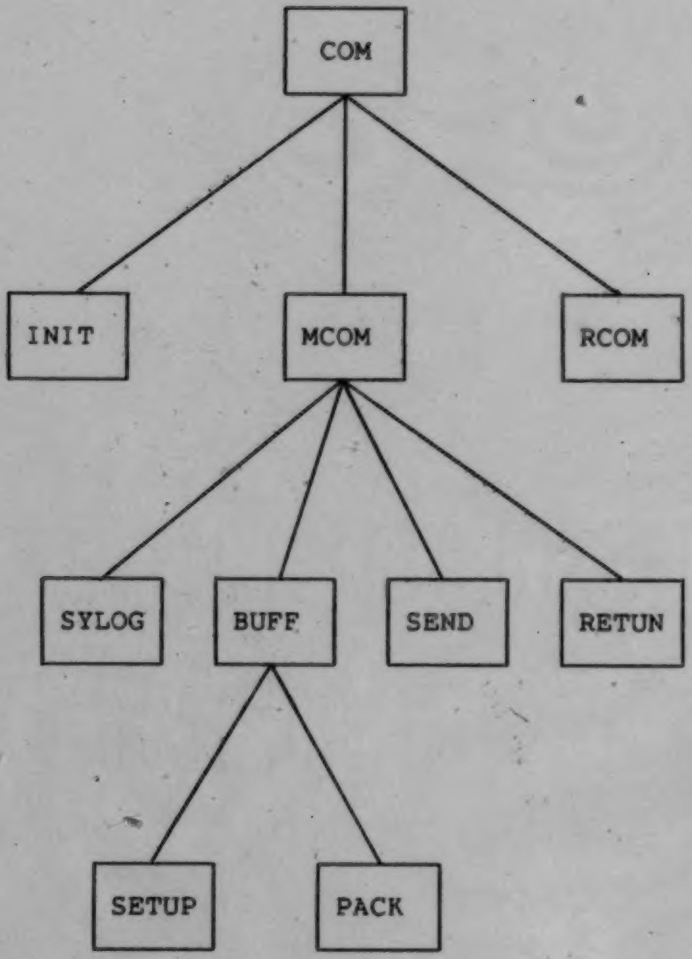
COM
INIT
MCOM
RCOM
SYLOG
BUFF
SEND
RETUN
SETUP
PACK

Figure 5.7 Detailed Module Layout for COM

```
(1) Read next request

(2) If request is from ACCES

        (A) Then send com. string to the node of the
               program initiating the com.

(3) Else If request is from RCOM

        (A) Then put com. string in receiving buffer of
               the appropriate com. number

        (B) If the buffer is full

            (a) Then check status flags

            (b) If the status indicate an acknowledgement is
                    to be sent

                (i) Then ack. to necessary nodes

            (c) Else If the status flags indicate the
                   com. request is complete

                (i) Then return the response
                   to the initiating program in this node
            (d) Else continue
        (C) Else return to (1)

(4) Else call SETUP

        - write com. string into log
        - sort string into separate lists according
           to nodes
        - make lists of nodes containing data
        - set up buffer for replys
        - send each list to the respective node

(5) Return to (1)
```

Figure 5.8 Logic for MCOM Module

```
(1) Receive next message from the network

(2) If the message is a request for data
        Then send the com. string to ACCES

(3) Else send the com. string to MCOM

(4) Return to (1)
```

Figure 5.9 Logic for RCOM Module

the requests. It does have one option to go to the INIT sub-module to initialize all the communication links. This option is only used when the program is started up for the first time.

In the normal mode of operation the request is placed in a stack. The request that this program receives may be requesting that data is required or may be returning data in response to a request. The stacking is accomplished using the same utilities as ACCES in performing this function. The program takes the request off the stack and converts it from one communication string to a separate communication string for each processor involved in the communication. The program then sends the requests (communication strings) to the respective processors as well as sending them all to a monitoring (MCOM) program. Once this is completed the COM module is placed in a wait state for the next request.

INIT

This module establishes the communication routes and links between the various nodes. It must be entered whenever there is a data link failure in order to establish a new connective path. This rerouting function will be developed with the development of the error sensing and correcting mechanisms.

RCOM

The RCOM sub-module is responsible for the function of receiving the messages from COM. RCOM acts as a slave to COM waiting for a request to be sent and responding only when asked. A request received by RCOM can initate two actions. The first action is to pass the communication string to the ACCES module for execution. This option is taken if the request is asking for processing. The second action is to pass the communication string to MCOM. This option is taken if the request is the response to a subsequent communication initiated at the local node (node where module resides). This decision as to which module to send the communication string to is made by analysing the control words in the first six words of the communication string. Once the string has been passed on the program returns to the wait state for activation by a new request.

MCOM

MCOM monitors the communications in progress. It provides a marshalling area for the response of the processors to a request that involves a number of nodes. Once all the processors involved in one request have responded the program passes the combination of the responses to the program initiating the request. Part of the function of the module is to monitor the duration of any one request The communication request is cancelled if it takes too long for processing. This prevents one request from tying up the communication channels and the memory space in the stacks.

The module operates by receiving the initial communication string from the COM module. It establishes a buffer for the responses to the generated requests, creates a checklist of the processors that must respond and starts a clock for excessive execution time. Once all the processors have responded MCOM sends all the responses to the initiating program and clears the buffer.

Another function of the MCOM is to ensure the integrity of the communication. This is of particular importance in the case where "writes" are being made to databases in a number of CPU's. To accomplish this the module must perform a series of operations similar to that of a secure write to a database. The module is responsible for co-ordinating all the acknowledgments from the ACCES modules of the processors involved. A detailed illustration of the protocol used is shown in Figure 4.4 (page 82).

The detailed design of the other modules mentioned in the introduction to this chapter are not included for two reasons. First, in most cases, a existing module is just expanded rather than being entirely redesigned. Secondly, the modules whose design was discussed are those concerned with the major objective of this thesis. The expanding of existing modules just facilitates the operation of the remainder of the system.

The sixth chapter deals with the testing and analysis of the performance of the process control system. This includes the individual modules as well as the system as a whole.

6. PROGRAM TESTING

The testing phase in a software product development demonstrates the actual success of the design and implementation. This is the first instance where the performance of the program actually can be compared to the specifications. Testing might show that modifications are required or that the whole approach must be reconsidered. Therefore, the testing procedure must be started as soon as possible to reduce the amount of unproductive work.

The testing procedure has to be methodical and logical to ensure the system performs as designed and to minimize the testing time. The design of the program involved the successive decomposition of the original problem until it was made up of small well defined modules (top down design). A "basic module" refers to any module that does not require further decomposition to achieve the design objectives while still requiring only one page of code. A bottom-up testing procedure takes the reverse approach, starting with the smallest, basic module and then working up the hierarchy. The testing procedure works on the principle that once a module is tested and passed, then and only then, can it be used as a block to build a larger program. This describes the "bottom-up" testing technique (28).

The testing phase can be divided into operability testing and performance testing. The former is concerned with determining if the functional objectives are met and the latter is concerned with the efficiency with which the module performs the functions. The operability testing step will be discussed first.

6.1 Operability Testing

This term refers to the step where the module is tested for the operability of the design functions and its response in a "bad" environment (incorrect inputs, missing resources, etc). During this phase the module is rigorously exercised. The degree of the thoroughness of the testing is reflected later in the life of the system since it is the bugs that are not found and eliminated during this phase that will appear and cause a system failure. Each stage of the testing procedure will be discussed generally. The specifics of the testing that was carried out on the enhancements of the DISCO program will follow the general discussion.

6.1.1 Functional level

The lowest level of testing looks at the subroutines making up the basic functions as identified on the final design diagram. A basic function is one which does not require a further division into sub-modules to execute its objectives while still remaining only one page of code. These modules should be designed to follow the form of Source/Transform/ Sink (28). The testing of each module varies the input strings (Source) and then looks at the output strings (Sink) generated by the executing module (Transform). Before the module is passed as operational all combinations of the input are tried to ensure that none produce an unexpected result. This brute force analysis is only possible since the number of input combinations is small.

The testing of the modules design for the enhancements followed this procedure exactly. Each module was designed in the form of Source/Transform/Sink so the varying of the input (to the module) strings tested all the functions.

6.1.2 Program Testing

The next step of the testing phase checks the operation of a number of modules as a larger functional group or program. This tests the communication between the modules and the order of execution. Several different operating conditions that a module might encounter are tested. This is accomplished by deliberately inserting errors in the various sub-modules and observing the net response. The number of variations attempted depends upon the timing requirements of the developmental process and the importance of the initial reliability. Since reliability is a critical feature in a process control system this step takes a considerable time. The procedure of building up the system (integration) is continued until a complete version of the product results.

This stage of testing resulted in the finalization of the ACCES and the COM modules. These were now ready to be used in the existing DISCO program. In the testing process, statements were included to illustrate the flow of the program. The program documentation (44) contains copies of complete runs of the ACCES and COM modules.

6.1.3 Application Testing

For a process control system a test job stream will be made up of the following tasks which are typical in an actual process environment.

(1) A Simple PID Loop

This case will constitute 80 to 90 percent of the work load of the system. Any small inefficiencies will be located and eliminated since they will have such a large effect on the performance.

(2) Split Cascade Control Problem

This type of job tests the systems handling of frequent updates to the database. This type of test job exercises the database managment, checking that the security measures are adequate and operate efficiently.

The term "split" in the title refers to the fact that the two entities in the cascade control system are running in different nodes. Such an application is unlikely in an actual process environment but it presents a severe test of the system communication protocols. The primary parameter tested here is the speed of communication.

(3) Sequencing Application

A sequencing application is one type of control problem where the status of the system must always be brought back to the state prior to a fault. An example of this is a mixing sequence where ingredients are added at certain times. If a failure occurs and the system is not brought back to the last step in the sequence then the whole batch must be discarded since the content would be unknown. Thus this job will test the recovery mechanism that is necessary for a process control system.

(4) Supervisory Control

Supervisory control is a good test for the network communication protocols since it typically involves updates to values in a number of nodes simultaneously. The DISCO program must ensure that all values are received simultaneously and correctly.

The application testing stage on the enhancements was limited.

In summary, the result of the work performed for this thesis accomplished the following. The integrity of the database is improved. The database is now isolated from the rest of the system and there is a definite interface. The interface program guarantees time consistency and performs some simple error checks. The recovery mechanism is not developed at this time so the reconstruction of the data base after a failure could only be approximated. The testing of the distributed operation was also limited since there is only one processor that has the capability of entering process data into the system. As a result the testing of the multi-processor operation was simplified to a test of the communications protocols.

With the successful check-out of the system on a jobstream made up of a combination of the above applications the system is ready for process control use in a plant. The actual ratio of the various jobs in the test jobstream would be determined by analysing the requirements of each particular process plant. In the testing of commercial system software a number of jobstreams would be used that constitute a cross section of the various possible uses of the system.

6.2 Performance Evaluation

System performance is a measure of how well a computer executes the task it was designed for. The criteria of measurement is as variable as the applications the computers are used for since each application requires a specific combination of features. A list of some of the criteria of performance indicies are accuracy, reliability, maintainability, expandability, flexibility and operation cost. A critical factor in a process control computer is the time base consistency. This is a reflection of the overall system load and the efficiency with which the system processes the load. System load on a process control machine is the number of applications required to be executed each processing interval. The execution time of any activity is broken into system overhead and the actual processing time required by the activity itself. The latter value is uncontrollable since it is dependent on each routine so it is not considered in the performance evaluation. A major performance concern is the actual system overhead required to set up the various routines. The means of attaining an evaluation of the various system modules is to measure the execution time of each and the frequency of use. This identifies the critical areas for special optimization.

The procedure for the performance evaluation of the enhancements followed the steps outlined in the operability testing phase. (bottom up) The execution time of the basic functions are evaluated first. This procedure was hampered by the fact that the finest gradation of the time of the trial hardware system was ten milliseconds. As a result all the basic functions could only be identified as taking less than this interval.

The second phase of the evaluation was the execution time of the functions made up of many modules. It was at this point that one of the original constraints had a significant effect.

The constraint that "the standard functions provided by the vendor must be used" contributed significantly to the execution time of the larger modules. The Hewlett Packard architecture limits the size of any one program. This dictates that the overall system must be segmented into separate programs. The segmentation procedure was fairly simple since each program was made up of modules that were unique, separate and independent. However, the communication between the programs within one CPU used the standard functions provided by the HP software. An analysis of the execution time of these standard communication functions found it to be quite substantial. This fact required the altering of the design of the system to minimize the interprogram communication. The resulting mode of operation grouped a number of control activities into "blocks". The block was configured such that it behaved like a transaction. This ensured the preservation of the integrity of the overall system.

The same constraint affected the design of the interprocessor communication. The standard functions made frequent (every interval) interprocessor communication impossible if the time base was to be maintained constant. This characteristic also results in a large difference between the time each input was sampled for data apparently sampled at the same instant (skew). As a result of overhead and possible data skew (exampl[illegible] in Appendix B.2.1.2) , interprocessor communication should only be used in applications that have a large execution interval (time between successive executions).

The execution time of the ACCES and COM functions are contained in Table 6.1 and 6.2 respectively. It should be noted that the execution time of the COM module refers to the time required to pass a message from one CPU to the recipient. This does not mean the computers are tied up

Function	# wd. access	# access request	Ave.time	Max.time
Read	1	1	21	50
	1	8	25	60
	8	1	21	60
Simple write	1	1	6	10
	1	8	15	40
	8	1	9	40
Secure write	1	1	179	6150
	1	8	253	5740
	8	1	185	5960

- times in milliseconds

Table 6.1 Execution Times for ACCES Functions

# Wd Sent	Ave. Time	Max. Time
64	78	460
256	284	450
512	604	990

- times in miliseconds

Table 6.2 Execution Time for COM Function

totally for communication for this length of time. A larger network is required to evaluate the time to send a message to a number of processors.

The final phase of the performance evaluation involves the system operating on a test job stream made up of the applications mentioned in the operability testing section. The parameters measured here include data skew, total overhead, and the maximum number of any one type of application. A detailed study using specialty tools are required to get an accurate and indepth evaluation of the system performance. Such a study would be performed by a systems analyst. This phase of the performance evaluation can only be done once the system is installed. As a result no evaluations of this type were made on any part of the DISCO program.

The last chapter of the thesis summarizes the goals of the investigation, discussed in chapter 1 to 4, draws conclusions from the data gained from the analysis of the design, chapters 5 and 6, and finally, gives some areas for future development of the system as a useful and practical tool for process control.

7. SUMMARY, CONCLUSIONS AND FUTURE WORK

The work of this thesis dealt with the design and implementation of extensions to the DISCO system developed at the University of Alberta. The two main areas of concern were database security and integrity and distributed processing of a control application. This work brings the system closer to the ultimate aim of an operating environment that meets all the computer system needs of a modern process plant.

7.1 Contributions of the Thesis

The contributions to the field of computer control of industrial process are briefly summarized in the following list.

1) Analysis of computer requirements of a control system for a process plant (chap. 2)
2) Analysis of the security features required by a process control applicaton in comparison to typical data processing problems (chap. 3)
3) Analysis of the recovery mechanisms required for a process control system (chap. 3)
4) Analysis of the application of distributed processing in a process control system. The communication needs were identified and compared to typical distributed data processing systems (chap. 4 & Appendix B))

5) Design and implementation of programs to fullfil the data security and communication for distributed processing needs (chap. 5 & 6)

6). Design and inplementation of a modern alarm processing routine (Appendix A.1).

7.2 Conclusions and Recommendations

The points made in this section concern the design of the enhancements to DISCO for execution on a network of Hewlett Packard 1000 mini-computer. Generally, these conclusions will apply to all mini-computer based systems. Instances where the comments are perinent specifially to the Hewlett Packard machine will be noted.

One conclusion is only some of the data used in a practical process control system needs full integrity measures. This is possible since most of the values are updated every execution interval. Thus only changes which occur on variables that do not fit into this category must be audited. Changes which fall into this category are any that originated from the operators or engineering console and any made within the Activity that are not process inputs or outputs.

A second conclusion is that distributed activities with a small execution interval should be avoided since they represent a large amount of overhead. This makes the maintainance of the time base difficult. The method of avoiding this is to change the configuration of the system such that the data that is frequently updated all originates in the same node as the Activity.

A final observation follows from the results of the analysis of data gathered in the performance testing step of the testing phase. The points mentioned refer generally to all process control systems.

The values of performance data gathered largely consist of the execution speed of the standard high level calls offered with the system's software. These had to be used in the implementation to conform to the constraint that standard calls and a high level language be used. Such a constraint will affect the performance on any system since the standard calls are designed for general purpose use. The flexibility of these calls is paid for in execution time. The result is the overall design and operation must minimize the inter-module and inter-processor communication. The former is quite drastic on the HP machine since the module size is also limited by the architecture of the hardware.

The two contraints mentioned above force the following operating method. The initial request to ACCES by EXEC must get a number (block) of Activity Records. This technique increases the buffer sizes but reduces the inter-module communication.

This method of operation with the machine fully loaded as described above could be alleviated two possible ways. Both require the relaxing of some of the initial constraints.

The first method requires the eliminating of the constraints of the use of the standard high level vendor supplied software and a high level language in the implementation. This means a separate, efficient, machine specific program would be written to create the environment necessary for process control applications. This solution would easily meet the time base consistency requirements while still allowing for the separation of the activities (eliminating blocking). The disadvantage of such a solution is the length of time for the development of the system and the virtual cryptic code that results that is decipherable only by a systems analyst.

The other alternative splits the lower level control task into two parts implemented by separate hardware. In such a solution the fast computations and data acquistion would be carried out in dedicated microprocessors designed specifically for such a task. The mini-computer would act as a co-ordinator of the simple fast microprocessors and a sink for the transfer of bulk data for logging purposes collected over a number of intervals. In such a scheme, the mini-computer is the second level in a hierarchical system. This method requires the programming of the small microprocessors but this task is fairly simple since the constraints and objectives are very specific and inflexible.

The latter solution appears to be the best means of alleviating the loading problem on the system as designed. This corresponds to the direction that industrial firms in this area are taking. The mini-computer which is operating at the second level in the hierarchy would have no control activities with a small execution interval since these would all be contained in the fast dedicated processors in the lowest level of the control hierarchy. The result is an efficient real time operating system that co-ordinates fast dedicated microprocessors.

7.3 Future Work

This section outlines the modules that were identified but not considered in any detail during the design of the enhancements. In addition, the modules that have not been investigated are pointed out.

7.3.1 Future Modules

The one area in the real time portion of the program that needs expanding is the development of the error handling routines. In the completed version most errors would be corrected automatically. Even the errors which are not automatically corrected should not cause a system failure but simply shut off the Activity that caused it and generate a report in the log. Another feature of the final version of the error handling routines is the automatic reassignment of the network loading in the event of the failure of one node. At the moment, the ERRHA module simply reports the place of the occurence of an error and stops the program.

A second area that should be expanded is the procedure for detecting and handling network communication failures. Ultimately, new communication paths would be sought if a link was found to be faulty. An offline program that optimizes network performance would eventually be tied into the online reconfiguration routine.

The final area that is almost undeveloped at the present is the offline, interrupt loaded utilities that provide the interface between the operator and the real time system. This area is almost a separate topic outside of control.

7.3.2 Future Work

The future work can be divided into the system area and the control area. Areas of future investigation will be pointed out in both areas although only those in the control field are of interest to the chemical engineer.

The one system area that needs investigation is the optimum frequency for checkpointing distributed databases. All the Activity Record Tables in the various nodes of the network make up a distributed database. Work has been done to optimize the checkpointing of a single database. A distributed database changes the basis for the analysis so additional work is required.

An aspect of control work that has not been examined is the effect skewed data has on the stability of a control technique. Up to this point all work has assumed that all input data is sampled instantly. This assumption is valid for a centralized system. With the introduction of distributed control, this assumption is not necessarly valid and consequently, its effects must be studied.

A final area of interest in the control field is the use of variable execution rate control algorithms. These would be used at times of high system load to reduce the frequency of execution of some loops. The possibility of success in this area is unlikely since the extensive work in the automatic tuning of loops has only met with minimal success. A variable execution rate algorithm would have to change the tuning parameters to match the change in the execution frequency.

The actual programs written to implement the enhancements are contained in a separate volume (44) and are not considered as an official part of this thesis.

BIBLIOGRAPHY

1. A. Avizienis, "Fault-Tolerance: The Survival Attribute of Digital Systems", Proceedings of the IEEE, Vol. 66,#10, pp. 1109- 1125, 1978.

2. C.T. Baker, "Logical Distribution of Applications and Data", IBM Systems Journal, Vol. 19,#2, pp. 171-191,1980.

3. P.A. Bernstein, D.W. Shipman, W.S. Wong, "Formal Aspects of Serializability in Database Concurrency Control", IEEE Transactions on Software Engineering, Vol. SE-5,#3, pp. 203- 215, 1979.

4. L.A. Bjork, "Recovery Scenario for a DB/DC System", Proceedings of the ACM, pp. 142- 146, 1973.

5. L.A. Bjork, "Generalized Audit Trail Requirements and Concepts for Data Base Applications", IBM Systems Journal, Vol. 3, pp. 229- 245, 1975.

6. A. Brenek,"DISCO, A Distributed Computer Control System for Engineering Purposes", M.Sc. thesis, University of Alberta, 1979.

7. G.A. Champine, "Six Approaches to Distributed Data Bases", Datamation, pp. 69- 72, May,1977.

8. K.M. Chandy, J.C. Browne, C.W. Dissly, W.R. Uhrig, "Analytic Models for Rollback and Recovery Strategies in Data Base Systems", IEEE Transactions on Software Engineering, Vol. SE-1,#1, pp. 100- 110, 1975.

9. K.M. Chandy, "A Survey of Analytic Models of Rollback and Recovery Strategies", Computer, pp. 40- 47, May,1975.

10. S. Chang, "A Model for Distributed Computer System Design", IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-5,#6, pp. 344- 359, 1976.

11. W.W. Chu, P.P. Chen, "Tutorial: Centralized and

Distributed Data Base Systems", First International Conference on Distributed Computing Systems, IEEE Computer Society, Huntsville,Alabama, 1979.

12. R.M. Curtice, "Integrity in Data Base Systems", Datamation, pp. 64- 68, May,1977.

13. R.A. Davenport, "Data Base Integrity", Computer Journal,Vol. 19,#2, pp. 110- 116, 1975.

14. R.A. Davenport, "Distributed or Centralized Data Base", Computer Journal, Vol. 21,#1, pp. 7- 13, 1976.

15. C.T. Davies, "Recovery Semantics for a DB/DC System", Proceedings of the ACM, pp. 136- 140, 1973.

16. D. Davies, J.F. Wakerly, "Synchronization and Matching in Redundant Systems", IEEE Transaction on Computers, Vol. C-27,#6, pp. 531- 539, 1978.

17. M.J. Denhen, "Implications of Distribution on Control System Analysis and Design", Proceedings of IEE International Conference on Distributed Computer Control Systems, 1977.

18. D.G. Fisher, "A Distributed Computer Network for Process Control and Research Applications", Canadian Conference on Automatic Control, Montreal, 1979.

19. J.N. Gray, "Notes on Data Base Operating Systems", *Lecture Notes in Operating Systems*, 60, Springer-Verlag, pp. 393- 481, 1978.

20. R.C. Gunther, "*Management Methodology for Software Product Endineering*", Wiley-Interscience, New York, 1978.

21. J.P. Hayes, R. Yanney, "Fault Recovery in Multiprocessor Networks", Proceedings of the International Conference on Fault Tolerant Computing, Toulouse Fr., 1978.

22. A.L. Hopkins, T.B. Smith, "The Architectural Elements of a Symmetric Fault-Tolerant Multiprocessor", IEEE Transactions on Computers, Vol. C-24,#5, pp. 498- 505, 1975.

23. Infotech, "Principles of Recoverability in Transaction-processing Systems", Computer Communications, Vol. 1,#5, pp. 242- 252, 1978.

24. J.H. Lala, C.J. Smith, "Performance and Economy of a Fault-Tolerant Multiprocessor", National Computer Conference, pp. 481- 492, 1979.

25. M.E.S. Loomis, "Aspects of Data Base Design in a Distributed Network", Data Communications, pp. 83- 94, May,1980.

26. H. Lorin, "Distributed Processing: An Assessment", IBM Systems Journal, Vol. 18,#4, pp. 582- 603, 1979.

27. M.D. Mesarovic, D. Macko, Y. Takahara, "Theory of Hieracrchical, Multilevel, Systems", Academic Press, New York, 1970.

28. G.J. Myers, "Composite/Structured Design", van Nostrand Reinhold, New York, 1978.

29. D. Powell, J.C. Laprie, P. Romand, C. Aleonard, "RHEA: A System for Reliable and Survivable Interconnection of Real-Time Processing Elements", Proceedings of the International Conference on Fault Tolerant Computing, pp. 117- 122, 1978.

30. C.P. Pracht, "A Distributed Microprocessor-Based System Programmable by the Process Engineer", ISA Conference on Automatic Control, pp. 515- 517, 1976.

31. D.J. Rosenkrantz, R.E. Stearns, P.M. Lewis, "System Level Concurrency Control for Distributed Database Systems", ACM Transactions on Database Systems, Vol. 3,#2, pp. 178- 198, 1978.

32. D.J. Rypka, A.P. Lucido, "Deadlock Detection and Avoidance for Shared Logical Resources", IEEE Transactions on Software Engineering, Vol. 5,#5, pp. 465- 471, 1979.

33. E.F. Schagrin, "Evolution of a Distributed Control System", Transactions of the AIChE, pp. 72- 75,

June,1980.

34. Scherr, "Distributed Data Processing", IBM Systems Journal, Vol. 17,#4, pp. 324- 343, 1978.

35. J.D. Schoeffler, "Software Architecture for Distributed Data Acquistion and Control Systems", Seventh Triennual World Congress of IFAC, pp. 641- 648, 1978.

36. H.A. Schutz, "On the Design of a Language for Programming Real-Time Concurrent Processes", IEEE Transactions on Software Engineering, Vol. SE-5,#3, pp. 248- 255, 1979.

37. A.J. Shepherd, "A British Example of Distributed Computing", Datamation, pp. 87- 91, March,1978.

38. H.A. Spang, "Distributed Computer Systems for Control", IFAC Annual Conference, Tallin,USSR, pp. 47- 65, 1976.

39. M. Stonebraker, "Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES", IEEE Transactions on Software Engineering, Vol. SE-5,#3, pp. 188- 194, 1979.

40. L. Svobodova, "Reliability Issues in Distributed Information Processing Systems", Proceeding of the Annual International Conference on Fault Tolerant Computing, pp. 9- 16, 1979.

41. R.H. Thomas, "A Majority Consenses Approach to Concurrency Control for Multiple Copy Databases", ACM Tansactions on Database Systems, Vol. 4,#2, pp. 180- 209, 1979.

42. F.C.H. Waters, "Design of the IBM 8100 Data Base and Transaction Management System-DTMS", IBM Systems Journal, Vol. 18,#4, pp. 565- 581, 1979.

43. H.D. Wend, "On Hierarchical Control of Complex Technological Systems", Large Scale Systems, Vol. 1,#1, pp. 63- 75, 1980.

44. I.G. Reavill, "Documentation of ACCES and COM",DACS Centre publication, Department of Chemical Engineering, University of Alberta, 1981.

APPENDIX A: ALARM PROCESSING SEGMENT

An alarm is a signal that brings attention to an abnormal condition. In the process control field the abnormal condition is the plant deviating from a specified operating condition. The purpose of the design of this program is to provide an alarm function for the DISCO application program.

The traditional definition of an alarm in a process environment refers to a signal generated by a switch that is tripped when a single process measurement exceeds a specified limit. As a result, an alarm signalled the process in an abnormal condition. The resulting action of the operator, was a large change in a control variable causing a shock to be sent through the whole process. One means of preventing an upset in the process was to set the limit sufficiently close to the normal operating point that the operator can take evasive action. This resulted in many false alarms.

The entrance of computers into the alarm problem should revolutionize the whole concept. The computer brings the capability to analyse a number of process values at a single instance and use the result to predict future operating conditions. Thus the alarm function changes from one of signalling the occurance of an event (high level exceeded) to that of prediction of an event happening. This gives the operator a much greater chance of avoiding an upset without seriously bumping the process. If the use of the computer's logic is taken one step further, the alarm function not only predicts a deviation but also the control variables. This is only possible when a single course of action is required.

The tools needed by a modern computerized alarm function include access all process measurements, histories of the measurements, a pattern recognition routine and the high and low limit check test. The alarm segment attempts to provide these features.

A.1 Discussion of Functions

A.1.1 Identification

In the time prior to computer control, alarm identification meant sensing a process variable beyond a operating limit. Adding a computer in the control system broadens the scope immensely, since there is now a element that can perform arithmetic calculations and logical decisions and has memory for storage of past events. These new capabilities enable the extension of the alarm function to include the prediction and analysis of alarms. The ability to predict alarms is possible only as a result of the availability of all the process inputs and their past values. The alarm module of a control application program must have the resources and capability to predict the occurence of alarms.

A.1.2 Action

The addition of a computer in the control loop changes the "action" step from a simple notification to one of compensating for the deviation. The compensation is possible since all the manipulated variable are accesible and the computer has the speed and logic to calculate new values. The purpose of this "action" is to isolate the problem and minimize its effects so the operator can concentrate specifically on the malfunction.

A.2 Design of the Module

A.2.1 Identification

The identification section must decide using the process values and past histories whether the plant is in or is about to go into a operating state beyond the operating limits or the capabilities of the control strategy.

The first step of the process to make this decision is the retrieval of the new process inputs and any recent input values. In the DISCO program the actual sampling of the process along with any filtering or conversion of the values attained is done in other existing modules. This module simply retrieves the results from the other modules. The present version of DISCO does not have a universally (system wide) accesible log of the process histories so the alarm module must store any past values that are required. This need necesitates a working buffer that is maintained between executions of the alarm module.

Once the data is accessible the module cant start the testing procedure. There are two types of tests possible. The simple test is check if the plant is presently in an alarm state. This is accomplished by testing the present values of the inputs against specified limits. The second test involves a pattern recognition procedure to predict or analyse the occurence of an alarm. Since the type of function required is dependent upon the process involved the module uses a user written subroutine ALID# (where # is a variable ASCII character).

The decision as to which type of test is performed is controlled by the second word (status word) in the string passed to the module. Figure A.1 illustrates the layout of the complete communication string for a simple test. A complex text would have a similiar communication string structure up to part 6. Any words in the string beyond part 5 would be used for storage of passed values.

Once the testing step is complete the action step will take effect if the test results are positive.

A.2.2 Action

The first type of action possible is simply a notification by a message and/or a horn that the alarm test was affirmative. The second action, altering the plant operation, is performed by initiating a separate alarm program. The type of action required is identified in word 3 of the communication string.

```
LISTING OF BUILDER WORKING TABLE
======= == ======= ======= =====

TITLE GENERAL ALARM SEGMENT      IGR    80 149    57
 -99  PART 1: SEGMENT HEADER
   1LBSEGMENT ID.                                        8 I          13
   1  SEGMENT LENG.                                      8BI
 -99  PART 2: STATUS FLAGS CONTROLLING SEG.
   2LBALARM SEG.          ENABLE/DISABLE                 1 I           1
   2  STACK                  YES/NO                      1 I           0
   2  ALARM TYPE          (OPTION)                       2 I           0
   2  INDIRECT ADDR.         YES/NO                      1 I           0
   2  1ST LIMIT               HI/LO                      1 I           1
   2  2ND LIMIT               HI/LO                      1 I           0
   2  MUL. LIMITS         COMMON/SEP.                    1 I           1
   2  LIMIT TYPE            ABS. /DEV.                   1 I           1
   2  DEV. FROM           STPT. /LAST VAL.               1 I           1
   2  MUL. LIMIT TYPE (OPTION)                           2 I           0
   2  # OF ALARM PTS.                                    4 I           1
 -99  PART 3: ALARM ACTION
   3LBALTER R.T. COM.        YES/NO                      1 I           0
   3  WRITE MESS.            YES/NO                      1 I           1
   3  SCH. PGM.              YES/NO                      1 I           0
   3  SPARE (2 BITS)                                     2N
   3  ACTION TAKEN      (OPTION)                         3 I           0
   3  DIFF. ADDR. OF BIT                                 8 I           1
   4LBACTIVITY # (TO BE ALTERED)                        16 I      177777
   5LBLU # TO WRITE ALARM MESS.                          8 I           1
   5  ALARM MESS. #                                      8 I           1
   6LBPGM. TO BE SCHEDULED                              48 A       AAAAA
 -99  PART 4: ALARM STATUS VECTOR
   8LB1ST ALARM STATUS VECTOR                           16N
   9LB2ND ALARM STATUS VECTOR                           16N
 -99  PART 5: TAG # OF ALARM POINTS
9999LBTAG # OF ALARM POINTS                             16 1           1
      TAG #                                             16 I
 -99  PART 6: LIMITS TO BE CHECKED
  -1LBDEADBAND                                          16 I          10
9999LB1ST LIMIT                                         16 1           1
      1ST LIMIT                                         16 I
9999LB2ND LIMIT                                         16 1           0
      2ND LIMIT                                         16 I
 -99  PART 7: DATA WORDS
9999LBSTPT. /LAST VAL.                                  16 1           0
      STPT. /LAST VAL.                                  16 I          50
 -99  PART 8: END OF TABLE
   EN
```

Figure A.1 Structure of Alarm Segment Communication String

A.3 Discussion of Design

The alarm module design has the capabilities to perform simple tests as It also has the "hooks" to include process specific programs that perform complex pattern recognition. The output from the module can write simple messages or initialize a complex shutdown (or equivalent) scheme.

One problem with alarm identification and subsequent action is each process requires a unique test and response. This fact makes the implementation of the complex schemes very labour intensive. However, with increased automatization of process plants such alarm mechanisms will be necessary. The computer may eventually evolve to strictly an alarm process, leaving the actual control to the new generation of instrumentation.

A second problem with computer alarming is the inability of a computer to work relatively. An example of this is an input pattern may be similar to the pattern which created an alarm condition will not trigger the action since the patterns are not identical. A solution for this type of problem may come from the area of "fuzzy logic".

APPENDIX B: ADVANTAGES AND DISADVANTAGES OF USING A DISTRIBUTED COMPUTER NETWORK

B.1 Advantages of Using a Distributed System

B.1.1 Reliability, Recoverability and Stability

The most important feature required from any equipment performing control is consistency. This translates into reliability, stability and recovery.

Processors in a distributed system are more reliable and stable for a number of reasons. Simplicity is the first. A large expensive mainframe used in a centralized system must perform all that is demanded of it. This dictates that the one machine must have all the possible functional capabilities. The complexity of such systems naturally leads to hidden design bugs that will only be found through varied use. Also the various functions require a lot of additional circuitry which creates a greater possbility of hardware failure.

Stability is affected when the system is updated with new releases. A large mainframe is always having modifications made to its hardware and software in attempts to prevent it from becoming prematurely obsolete. Customers demand this to extend the payout period of their investment. The one problem in a constantly updated system is that additional bugs are acquired that must be found and resolved. This process takes time and increases the down-time. A distributed system is made up of small processors. Small processors, each with a specific purpose, are rarely updated. Rather than update an existing processor with a marginal increase in performance it is usually easier to wait until a significant increase in performance is required and available and then completely replace that unit. The replacement unit is at least as reliable as its predecessor since the software and hardware are designed together. This approach is possible because the outlay for a new processor is fairly small.

The ability to recover from failures is a key feature favouring a distributed system. A central mainframe that encounters a major failure means that the entire system is out of service until it can be fixed. The only defence against this is to sense the fault in one of its modules ahead of time and isolate it so the computer can be kept operating at a reduced performance. This approach is only possible for minor failures because anything major will affect the whole system. A distributed system on the other hand can tolerate the failure of a processor by rebalancing the workload around the remaining processors in the network. The new network will operate at a reduced performance level. The detection mechanism and restructuring of the workload will require overhead. This dictates that all the machines will need a little better performance specification but this is easily justified by the fact that the system keeps operating at all times.

B.1.2 Communications

A totally centralized control system dictates that all the process transducers must be connected to one large computer system. This situation causes a large amount of overhead to be spent on the management of the process input and output. Attention must be paid to the balancing of the number of points among the limited number of high speed I/O ports or channels. The I/O sampling systems must be closely controlled and synchronized to enable all the points to be read each execution interval. Wire of special construction is often required to permit the high speed communication that is necessary. The one advantage with respect to communication of a centralized system is all the information is in one place and is accessible to all applications.

A distributed system has a much more manageable process I/O problem because each processor has a small number of points since the total for the plant are spread over a number of processors. However, this configuration means an application that needs inputs that are not connected to the host processor must get them through the communication network connecting the processors. The amount of interprocessor communication is considerably less than the process I/O of the centralized system for three reasons. First, data will not leave a processor unless it is required elsewhere. Thus a large percentage of the data transmission is eliminated since in most cases the data is used only by the processor located near the source. The second reason is each processor will format the data needed by another processor in such a way to optimize the communication, thus minimizing the time required for communication and for the receiving computer to spend in the waiting for communication state. The third reason concerns only the reporting function. Rather than sending all the data to the processor delegated with this task only the required averages are sent. Thus the amount of information transmitted is greatly reduced. All these facets leave the computers available to do useful work.

B.1.3 Economy of Dedicated Processors

It is a known fact that a program designed specifically for a particular function using a specified machine will be more efficient than a generalized program capable of doing a function, as well as variations of the function and executable on any machine. In the case of a centralized computer system it is necessary to provide all the variations of a function in order not to limit the computer's market. However, for process control, it is evident from the list of functions required in chapter 2 that the needs are very specific. A distributed system can be easily tuned to these needs since every unit can be programmed ultra-efficiently to accomplish one specific function. This reduces the size and cost of each processor. However, some degree of similiarity is required between processors otherwise one cannot be used to back up the other.

B.1.4 Incremental Growth and Flexibility

A large centralized control system is a fixed entity once it is installed. It is almost impossible to change the original specifications since all the components were designed to operate with one another. Thus once capacity or a specific function is not available a major reconfiguration and rebuild is necessary. This state is maintained until justification warrants the replacing of the whole machine. Traditionally, such a condition is postponed by initially purchasing a machine that has four or five times the needed capacity and performance.

A distributed system avoids this quandry because it is easily expanded in varying increments. If some additional capacity is required then one additional processor is added to the network. If a number of new features are necessary then a number of processors are added to accomodate the need. The one limitaion that must be watched is that the additional overhead required to maintain and backup the additional mode does not exceed the increase in performance. If such a case exists adding more processors simply degrades the system more. The alternate in this situation is to replace the existing processor with a more powerful one.

B.1.5 Installation

The final advantage of a distributed system is the installation can be in small blocks quantitizing the trouble-shooting and debugging required. Additional blocks to the network just extend the overall performance of the computer system but do not affect the portions already working.

B.2 Disadvantages Connected With a Distributed System

B.2.1 Performance

This section only refers to implementations when the data required for a given operation is located in a number of processors in the network. In such cases, particularly if the operation has a fast execution interval, the time required to retrieve data enters into the application (delay time due to communication enters into the timing of controller constants for a feedback loop).

B.2.1.1 Response Time

Response time refers to the overall time required to generate a response (usually involving a process) to a given stimulus (interrupts, process input data, etc). The length of the communication delay depends upon bandwith of the communication link between processors and the configuration of the network.

An example of how network configuration affects response time is illustrated with Figures 4.1 and 4.2. The configuration of Figure 4.2 will have considerably less communication time between Node 1 and Node 5 than Figure 4.1

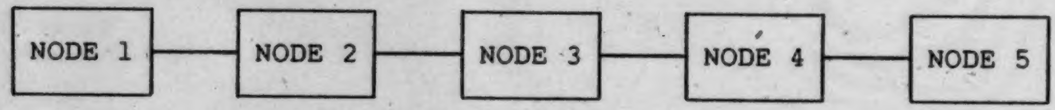
NODE 1
NODE 2
NODE 3
NODE 4
NODE 5

Figure 4.1 String Network

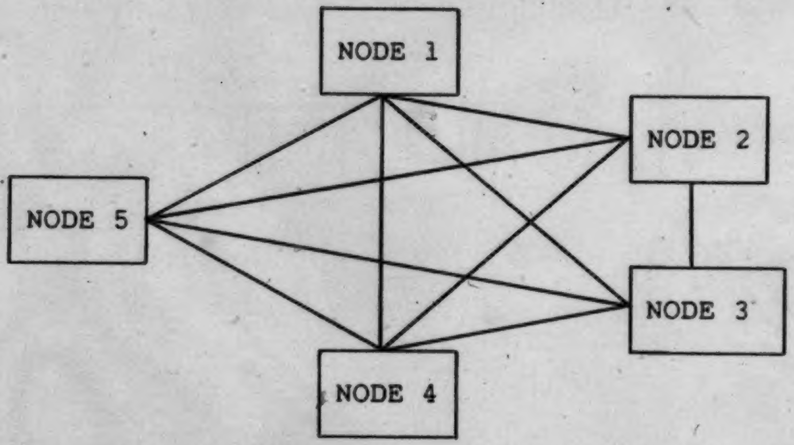
NODE 1
NODE 2
NODE 5
NODE 3
NODE 4

Figure 4.2 Fully Connected Network

because the former involves only two processors while the latter uses five. Each adds a delay since it must check error codes, translate routing instructions , generate new routing instructions and error codes and resend the data. The trade-off to make Figure 4.1 into Figure 4.2 is the wiring cost.

One solution to the reduced response time is to maintain a copy of all the data required by a processor in the processor. This solution creates the task of ensuring all copies of the data reflect the same information. This task requires a considerable amount of overhead, lengthening the time to execute each and every transaction.

There is a maximum time that data retrieval can take since the execution interval between successive calculations of the control action is fixed. The network architecture and the communication's protocols must be designed with this specification in mind.

B.2.1.2 Skew

Skew in data refers to the differences between the time the data was sampled. In a centralized system all the process data is brought into one place. The data skew is simply due to the cumulative effect of the the time required for the data to travel between any two points. Since this is so small this problem was ignored.

Skew in a distributed system is another matter since the communication time between processors enters into the problem. Communication time between processors can be significant since all the protocols for data checking must be established. Thus if two points are sampled at the same time in different processors and required for use in the same processor, the skew would be the time required to transmit the data from one processor to the other. This is compounded further if the data must make a number of hops through several processors to get from the process source to the node demanding the information. This discussion describes the minimum delay possible. Such a network would be made up of a series of processors acting as just multiplexers, distributed through the process plant gathering data for a single larger processor which performs all the calculations.

An exagerated example of this is illustrated in Figure 4.3 where process input point "1" of a control algorithm is sampled at 10:16:00 whereas input point "2" of the same algorithm is received at that time but was actually sampled at 10:15:30. The skew between the two points is 30 seconds. This could have a serious effect on the stability of the control loop since it has assumed that the points were sampled together. Possibly, the effect of the skew could be "tuned-out" by varying the parameters in the algorithm. This tuning process would require that the skew between data points is constant.

Data skew is greater in a network of computers where all have a number of tasks. In this case the computer would

TIME	NODE 1	NODE 2
10:15:10	- sample local value and update local value table	
10:15:15	- request data values from network	
10:15:20	- sample then update local value table	
10:15:30	- sample then update local value table	
10:15:35		- receive request for data value
10:15:36		- format return communication string
10:15:38		- send requested values
10:15:40	- sample then update local value table	
10:15:50	- sample then update local value table	
10:15:56	- receive requested updates for network data	
10:15:59	- enter new value in local value table	
10:16:00	- sample then update local value table	
10:16:05	- run application requiring local and network values	

- communication time 20 sec.

Figure B.3 Example of Commuication Delay Creating Skew

sample the data then preceed to execute other tasks. Once these were completed the process points would be sampled again. Thus the data at any one processor can be up one execution interval old. This initial time lag is then increased with the communication delay.

Data skew is inevitable in a distributed system with the process points entered at different nodes of the network. The only solution is to design the system such that control loops with a small execution interval have all the input points in one processor. This is the only case where the skew will be less than the smallest execution interval. In loops with a longer execution interval the skew of the input points can be neglected since the size of the skew becomes insignificant with respect to the sampling interval.

The need in these cases is that a maximum value for skew and response time be stated as a specification for a configuration of the network. If these values are too large for a particular application then it becomes the engineer's responsibility to deal with the control problem in another way.

B.2.2 Duplication of Function

A problem that is inevitable in a control system distributed largely on a geographical basis is a duplication of functions and hence of the corresponding computer hardware and software. With such a distribution criterion it is more efficient to have processors carrying out similar functions than to send all the data to a dedicated processor. Thus the processors are required to be larger and more general, performing a number of tasks. An example of this is detailed in section 4.3.2. This additional expense of the larger processors more than offsets the cost of the communications media to accomodate the traffic required in a system with strictly dedicated processors. Another advantage of the generalized approach is the transfer sequencing during a backup operation is much simpler.